МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

КУРСОВАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»
(наименование дисциплины)

Тема курсовой работы К_9 Моделирование провода корабля по фарватеру
(наименование темы)

Студент группы ИКБО-30-23 Мусатов Иван Алексеевич
(учебная группа) *(Фамилия Имя Отчество)* *(подпись студента)*

Руководитель курсовой работы доцент Быков А.Ю.
(Должность, звание, ученая степень) *(подпись руководителя)*

Консультант доцент Лозовский В.В.
(Должность, звание, ученая степень) *(подпись консультанта)*

Работа представлена к защите «31» мая 2024 г.

Допущен к защите «31» мая 2024 г.

Москва 2024 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра вычислительной техники

Утверждаю

Заведующий кафедрой _____
Подпись

Платонова О.В.
ФИО

«21» февраля 2024г.

ЗАДАНИЕ
На выполнение курсовой работы
по дисциплине **«Объектно-ориентированное программирование»**

Студент Мусатов Иван Алексеевич Группа ИКБО-30-23

Тема К_9 Моделирование провода корабля по фарватеру

Исходные данные:

1. Описания исходной иерархии дерева объектов.
2. Описание схемы взаимодействия объектов.
3. Множество команд для управления функционированием моделируемой системы.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:

1. Построение версий программ.
2. Построение и работа с деревом иерархии объектов.
3. Взаимодействия объектов посредством интерфейса сигналов и обработчиков.
4. Блок-схемы алгоритмов.
5. Управление функционированием моделируемой системы

Срок представления к защите курсовой работы: до «31» мая 2024 г.

Задание на курсовую работу выдал _____ (Быков А.Ю.)
Подпись *ФИО консультанта*
«21» февраля 2024 г.

Задание на курсовую работу получил _____ (Мусатов И.А.)
Подпись *ФИО исполнителя*
«21» февраля 2024 г.

Москва 2024 г.

ОТЗЫВ

на курсовую работу

по дисциплине «Объектно-ориентированное программирование»

Студент **Мусатов Иван Алексеевич** группа ИКБО-30-23
(ФИО студента) *(Группа)*

Характеристика курсовой работы

Критерий	Да	Нет	Не полностью
1. Соответствие содержания курсовой работы указанной теме	✓		
2. Соответствие курсовой работы заданию	✓		
3. Соответствие рекомендациям по оформлению текста, таблиц, рисунков и пр.	✓		
4. Полнота выполнения всех пунктов задания	✓		
5. Логичность и системность содержания курсовой работы	✓		
6. Отсутствие фактических грубых ошибок	✓		

Замечаний: нет

Рекомендуемая оценка: отлично

доцент Быков А.Ю.

(Подпись руководителя) *(ФИО руководителя)*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящая курсовая работа выполнена в соответствии с требованиями ГОСТ Единой системы программной документации (ЕСПД) [1]. Все этапы решения задач курсовой работы фиксированы, соответствуют требованиям, приведенным в методическом пособии для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [2-3] и методике разработки объектно-ориентированных программ [4-6].

При моделировании работы сложной информационной системы перед разработчиком неизбежно встает вопрос структуризации информации с точки зрения управляемости. Необходимо обеспечить работу системы так, чтобы все ее структурные элементы имели возможность взаимодействия друг с другом. Такое требование выполнимо в случае использования объектно-ориентированной парадигмы программирования. Представление элементов системы в качестве дискретных объектов позволяет организовать их иерархию и правила взаимодействия.

В данной курсовой работе рассматривается система, реализующая решение задачи проведения корабля по фарватеру. Целью работы является моделирование данной системы при помощи средств языка C++. Для реализации моделирования необходимо решить следующие задачи:

- Проанализировать поставленные требования к системе и выделить ее структурные компоненты;
- Изучить особенности объектно-ориентированного программирования на языке C++;
- Реализовать программное представление компонентов системы в виде классов;

- Через комплекс сигналов и обработчиков наладить взаимодействие компонентов;
- Посредством тестирования отладить получившуюся программу.

1 ПОСТАНОВКА ЗАДАЧИ

Надо моделировать работу системы, которая реализует решение задачи провода корабля по фарватеру. Система состоит из следующих элементов:

- акватория;
- корабль;
- локатор глубины;
- пульт управления со штурвалом.

Акватория имеет прямоугольную форму и разбита на квадраты. Для каждого квадрата задано значение его глубины. Этому разбиению по квадратам можно сопоставить матрицу (n x m). Где, n – количество строк, а m – количество столбцов. Значения n и m больше 10. Нумерация строк и столбцов начинается с 1. Первоначально корабль плывет вдоль первой строки квадратов и определяет глубину дна. Известно, что у фарватера есть только один вход. Корабль в рамках одного квадрата может развернуться влево или вправо, не более 90 градусов. Корабль по курсу движения может передвигаться в левую, в левую по диагонали, следующую, в правую по диагонали или в правую клетку. Локатор способен по курсу движения определить величину глубины в вышеперечисленных клетках. Для корабля дана минимально необходимая глубина. Основное направление курса движения от первой строки к строке n. Корабль на обратный курс не должен развернутся. Другими словами, не должен переместиться по ходу движения от клетки под номером строки i к строке под номером i-1. Всегда выбирается квадрат относительно квадрата расположения корабля, согласно его курсу движения, по максимуму глубины среди возможных квадратов, куда он может переместится.

Система ищет фарватер и проводит корабль по него. Фарватер считается пройденным, если корабль достиг клетку с номером строки n. Надо отметить, что такой фарватер всегда существует.

Акватория моделируется посредством прямоугольной целочисленной матрицы. Элемент матрицы соответствует квадрату, а его значение глубине квадрата. У корабля есть характеристика курса движения, координаты расположения и минимальная допустимая глубина.Время функционирования системы разбита на интервалы и это моделируется посредством тактов. В рамках каждого такта элементы системы выполняют предопределенные действия. Инициирование действий моделируется посредством запросов (сигналов).

Элементы системы моделируются посредством следующих объектов:

1. Объект «система». Обеспечивает отработку тактов.
2. Объект для чтения данных и команд. Объект считывает данные для подготовки и настройки системы. После чтения очередной порции данных для настройки или данных команды, объект выдает сигнал с текстом полученных данных. Все данные настройки и данные команд синтаксический корректны. Каждая строка данных или команд соответствует одному такту.
3. Объект пульта управления. Моделирует работу лоцмана, капитана и штурвала. Отрабатывает поступающие данные и команды. Объект вырабатывает решение и выдает соответствующий сигнал или сигналы. Выдает команду локатору для определения глубин в квадратах по курсу движения, выбирает допустимое направление движения, устанавливает штурвал и дает команду на движение корабля. Готовит данные исходя из состояния системы. Действия выполняется в рамках одного такта.
4. Объект, моделирующий акваторию. Содержит информацию относительно размеров акватория. Количества строк и столбцов квадратов. Относительно каждого квадрата значение его глубины. По запросу от локатора, посредством сигнала возвращает величину глубины, указанной в тексте запроса квадрата или нескольких квадратов.

5. Объект, моделирующий корабль. Содержит информацию о курсе движения и координате расположения. Координата соответствует номеру строки и столбца. Выполняет перемещение корабля по квадратам согласно направлению. Корабль может переместится в одно из соседних квадратов. За одни такт корабль может развернуться в рамках одного квадрата не более чем на 90 градусов. Принимает сигнал (команду) от пульта управления.
6. Объект, моделирующий локатор. Получает команду от пульта управления. Опрашивает квадраты акватории исходя из расположения и курса корабля. По сигналу возвращает полученные данные относительно глубин допустимых для перемещения квадратов.
7. Объект для вывода состояния или результата команды системы на консоль. Текст для вывода объект получает по сигналу от других объектов системы. Каждое присланное сообщение выводиться с новой строки.

Написать программу, реализующую следующий алгоритм:

1. Вызов метода объекта «система» build_tree_objects ( ).
    1.1. Построение дерева иерархии объектов. Характеристики объектов вводятся.
        1.1.1. Установка связей (сигнал-обработчик) и приведение части объектов в состоянии готовности, для обеспечения ввода данных и настройки системы.
        1.1.2. Выдача сигнала объекту чтения для ввода данных.
        1.1.3. Отработка операции загрузки данных.
    1.2. Установка связей сигналов и обработчиков между объектами.
2. Вызов метода объекта «система» exec_app ( ).
    2.1. Приведение всех объектов в состояние готовности.
    2.2. Цикл для обработки такта функционирования системы.
        2.2.1. Выдача необходимых сигналов.

2.2.2. Отработка взаимодействия объектов и необходимых алгоритмов.

2.3. После прохода корабля по фарватеру завершить работу.

Исходное расположение корабля координата (1,1) и курс направлен вдоль первой строки.

Все приведенные сигналы и соответствующие обработчики должны быть реализованы. Запрос от объекта означает выдачу сигнала. Все сообщения на консоль выводятся с новой строки.

В набор поддерживаемых команд добавить команду «SHOWTREE» и по этой команде вывести дерево иерархии объектов системы с отметкой о готовности и завершить работу системы (программы).

1.1 Описание входных данных

Ввод информации об акватории.

```
«количество строк» «количество столбцов» «минимальная глубина»
```

Со второй строки ввод информации о глубинах квадратов построчно по «количеству столбцов» штук.

```
«значение глубины» . . . . .
. . . . . .
```

Пример ввода:

```
20 20 15
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
10 10 11 12 20 13 10 10 10 25 10 10 10 10 10 10 10 10 10 10
10 10 10 10 35 10 10 10 27 10 10 10 10 10 10 10 10 10 10 10
10 10 10 36 10 10 10 10 30 31 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 32 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 25 21 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 27 17 21 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 33 10 36 37 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 32 33 34 33 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 35 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 36 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 37 10 10 10 10 10
```

```
10 10 10 10 10 10 10 10 10 10 10 10 10 38 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 40 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 41 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 43 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 44 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 45 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 46 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 47 10 10 10 10 10
```

1.2 Описание выходных данных

После завершения построения дерева иерархии объектов и загрузки исходных данных отображается:

```
Ready to work
```

Вывод найденного фарватера. Шаблон вывода информации о найденном фарватере:

```
Fairway «координата 1» «координата 2» . . . .
```

Координаты выводятся по шаблону:

```
(«номер строки», «номер столбца»)
```

Сообщение завершения работы системы:

```
Turn off the ATM
```

Пример вывода:

```
Ready to work
Fairway (1, 10) (2, 10) (3, 9) (4, 10) (5, 11) (6, 10) (7, 9) (8, 10) (9, 11)
(9, 12) (10, 13) (11, 14) (12, 15) (13, 15) (14, 15) (15, 15) (16, 15) (17,
15) (18, 15) (19, 15) (20, 15)
Turn off the ATM
```

2 МЕТОД РЕШЕНИЯ

Для решения задачи используется:

- объект system класса System предназначен для организации связи объектов через дерево иерархии;
- объект *reader класса Reader предназначен для чтения данных и команд;
- объект *display класса Display предназначен для вывода состояния или результата команды системы на консоль;
- объект *area класса Aquatory предназначен для моделирования акватории;
- объект *vessel класса Vessel предназначен для моделирования корабля;
- объект *control класса ControlBoard предназначен для моделирования работы лоцмана, капитана и штурвала;
- объект *locator класса Locator предназначен для моделирования локатора;
- объект стандартного потока ввода данных с клавиатуры cin;
- объект стандартного потока вывода данных на экран cout;
- условный оператор if else;
- оператор множественного выбора switch case;
- оператор цикла с предусловием while;
- оператор цикла с параметром for.

Класс Base:

- функционал:
    - метод ~Base — деструктор.

Класс System:

- функционал:
    - метод System — параметризированный конструктор;
    - метод buildTree — построение дерева иерархии;
    - метод run — основная логика работы приложения;

- метод order — метод сигнала для объектов системы;
- метод cmdProcesser — метод обработчика команд пользователя;
- метод ~System — деструктор.

Класс Reader:

- функционал:
  - метод Reader — параметризированный конструктор;
  - метод readln — обработчик команд системы;
  - метод read — сигнал со считанной информацией;
  - метод ~Reader — деструктор.

Класс Display:

- функционал:
  - метод Display — параметризированный конструктор;
  - метод print — обработчик запросов о выводе данных;
  - метод ~Display — деструктор.

Класс Aquatory:

- свойства/поля:
  - поле номер текущей строки:
    - наименование — curLine;
    - тип — int;
    - модификатор доступа — private;
  - поле число строк:
    - наименование — n;
    - тип — int;
    - модификатор доступа — public;
  - поле число столбцов:
    - наименование — m;
    - тип — int;

        - модификатор доступа — public;
    - поле матрица глубин:
        - наименование — area;
        - тип — int**;
        - модификатор доступа — public;
- функционал:
    - метод Aquatory — параметризированный конструктор;
    - метод setProperties — обработчик для установки параметров;
    - метод setLine — обработчик для установки глубин;
    - метод returnDepth — сигнал локатору;
    - метод getSignal — обработчик сигналов локатора;
    - метод ~Aquatory — деструктор.

Класс Vessel:

- свойства/поля:
    - поле хранение фарватера:
        - наименование — fairway;
        - тип — string;
        - модификатор доступа — private;
    - поле координаты текущего расположения:
        - наименование — cur;
        - тип — Cell*;
        - модификатор доступа — public;
    - поле координаты входа в фарватер:
        - наименование — entrance;
        - тип — Cell*;
        - модификатор доступа — public;
    - поле направление движения:

        - наименование — course;
        - тип — Direction;
        - модификатор доступа — public;
    - поле минимально допустимая глубина:
        - наименование — minDepth;
        - тип — int;
        - модификатор доступа — public;
- функционал:
    - метод Vessel — параметризированный конструктор;
    - метод printFairway — сигнал для вывода фарватера;
    - метод setProperties — обработчик для установки параметров;
    - метод move — обработчик команды пульта управления;
    - метод ~Vessel — деструктор.

Класс ControlBoard:

- свойства/поля:
    - поле максимальная доступная глубина:
        - наименование — maxDepth;
        - тип — int;
        - модификатор доступа — private;
    - поле режим работы пульта:
        - наименование — entranceFound;
        - тип — bool;
        - модификатор доступа — private;
- функционал:
    - метод ControlBoard — параметризированный конструктор;
    - метод order — сигнал-команда;
    - метод getDepth — обработчик локатора;

- метод ~ControlBoard — деструктор.

Класс Locator:

- свойства/поля:
  - поле текущее рассматриваемое направление:
    - наименование — curDir;
    - тип — Direction;
    - модификатор доступа — private;
- функционал:
  - метод Locator — параметризированный конструктор;
  - метод locate — сигнал локатора;
  - метод returnDepth — сигнал возврата исследованной глубины;
  - метод receiveCmd — обработчик команд пульта управления;
  - метод receiveDepth — обработчик ответа от акватории;
  - метод ~Locator — деструктор.

Таблица 1 – Иерархия наследования классов

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
1	Base			Базовый класс для реализации дерева иерархии	
		System	public		2
		Reader	public		3
		Display	public		4
		Aquatory	public		5
		Vessel	public		6
		ControlBoard	public		7
		Locator	public		8
2	System			Класс для реализации связи элементов	

№	Имя класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер
				системы	
3	Reader			Класс для реализации чтения данных	
4	Display			Класс для реализации вывода данных	
5	Aquatory			Класс для представления акватории	
6	Vessel			Класс для моделирования корабля	
7	ControlBoard			Класс представления пульта управления	
8	Locator			Класс представления локатора	

3 ОПИСАНИЕ АЛГОРИТМОВ

Согласно этапам разработки, после определения необходимого инструментария в разделе «Метод», составляются подробные описания алгоритмов для методов классов и функций.

3.1 Алгоритм деструктора класса Base

Функционал: Стандартный деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 2.

Таблица 2 – Алгоритм деструктора класса Base

№	Предикат	Действия	№ перехода
1		Объявление и инициализация нулем целочисленной переменной i; переменной size - размером поля-коллекции successors	2
2	i < size	Удаление объекта по адресу поля successors под индексом i	3
			4
3		Увеличение i на 1	2
4		Присвоить i нулевое значение; size - размер поля-коллекции bonds	5
5	i < size	Удаление объекта по адресу поля bonds под индексом i	6
			∅
6		Увеличение i на 1	5

3.2 Алгоритм конструктора класса System

Функционал: Параметризированный конструктор.

Параметры: Base* ancestor - указатель на головной объект.

Алгоритм конструктора представлен в таблице 3.

Таблица 3 – Алгоритм конструктора класса System

№	Предикат	Действия	№ перехода
1		Делегирование конструктору базового класса Base параметра ancestor	∅

3.3 Алгоритм метода buildTree класса System

Функционал: Построение дерева иерархии.

Параметры: нет.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода buildTree класса System

№	Предикат	Действия	№ перехода
1		Вызов у текущего объекта метода rename с параметром "System"	2
2		Создание объекта reader класса Reader с параметрами конструктора: указателем на текущий объект и строкой "Reader"	3
3		Создание объекта display класса Display с параметрами конструктора: указателем на текущий объект и строкой "Display"	4
4		Создание объекта area класса Aquatory с параметрами конструктора: указателем на текущий объект и строкой "Aquatory"	5

№	Предикат	Действия	№ перехода
5		Создание объекта vessel класса Vessel с параметрами конструктора: указателем на текущий объект и строкой "Vessel"	6
6		Создание объекта control класса ControlBoard с параметрами конструктора: указателем на объект vessel и строкой "ControlBoard"	7
7		Создание объекта locator класса Locator с параметрами конструктора: указателем на объект vessel и строкой "Locator"	8
8		Вызов у текущего объекта метода setState с параметром 1	9
9		Вызов у объекта reader метода setState с параметром 1	10
10		Вызов у объекта area метода setState с параметром 1	11
11		Вызов у объекта vessel метода setState с параметром 1	12
12		Вызов у объекта display метода setState с параметром 1	13
13		Вызов у текущего объекта метода connect с параметрами: сигналом order класса System, указателем на объект reader , обработчиком readln класса Reader	14
14		Вызов у объекта reader метода connect с параметрами: сигналом read класса Reader, указателем на объект area, обработчиком setProperties класса Aquatory	15
15		Вызов у объекта reader метода connect с параметрами: сигналом read класса Reader, указателем на объект vessel, обработчиком	16

№	Предикат	Действия	№ перехода
		setProperties класса Vessel	
16		Вызов у текущего объекта метода emitSignal с параметрами: сигналом order класса System, строкой "READ"	17
17		Вызов у объекта reader метода disconnect с параметрами: сигналом read класса Reader, указателем на объект area, обработчиком setProperties класса Aquatory	18
18		Вызов у объекта reader метода disconnect с параметрами: сигналом read класса Reader, указателем на объект vessel, обработчиком setProperties класса Vessel	19
19		Вызов у объекта reader метода connect с параметрами: сигналом read класса Reader, указателем на объект area, обработчиком setLine класса Aquatory	20
20		Объявление и инициализация нулем целочисленной переменной i	21
21	i < поле n объекта area	Вызов у текущего объекта метода emitSignal с параметрами: сигналом order класса System, строкой "READ"	22
			23
22		Увеличение i на 1	21
23		Вызов у объекта reader метода disconnect с параметрами: сигналом read класса Reader, указателем на объект area, обработчиком setLine класса Aquatory	24
24		Вызов у объекта control метода connect с параметрами: сигналом order класса ControlBoard, указателем на объект locator, обработчиком	25

№	Предикат	Действия	№ перехода
		receiveCmd класса Locator	
25		Вызов у объекта control метода connect с параметрами: сигналом order класса ControlBoard, указателем на объект vessel, обработчиком move класса Vessel	26
26		Вызов у объекта locator метода connect с параметрами: сигналом returnDepth класса Locator, указателем на объект control, обработчиком getDepth класса ControlBoard	27
27		Вызов у объекта locator метода connect с параметрами: сигналом locate класса Locator, указателем на объект area, обработчиком getSignal класса Aquatory	28
28		Вызов у объекта area метода connect с параметрами: сигналом returnDepth класса Aquatory, указателем на объект locator, обработчиком receiveDepth класса Locator	29
29		Вызов у объекта reader метода connect с параметрами: сигналом read класса Reader, указателем на текущий объект, обработчиком cmdProcesser класса System	30
30		Вызов у текущего объекта метода connect с параметрами: сигналом order класса System, указателем на объект display, обработчиком print класса Display	31
31		Вызов у объекта vessel метода connect с параметрами: сигналом printFairway класса Vessel, указателем на объект display, обработчиком print класса Display	32

№	Предикат	Действия	№ перехода
32		Вызов у текущего объекта метода emitSignal с параметрами: сигналом order класса System, строкой "Ready to work"	∅

3.4 Алгоритм метода run класса System

Функционал: Основная логика работы приложения.

Параметры: нет.

Возвращаемое значение: int - корректность выполнения программы.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода run класса System

№	Предикат	Действия	№ перехода
1		Вызов у объекта, найденного в результате вызова метода find с параметром "Locator", метода setState с параметром 1	2
2		Создание объекта area класса Aquatory и его инициализация результатом вызова метода find с параметром "Aquatory"	3
3		Создание объекта vessel класса Vessel и его инициализация результатом вызова метода find с параметром "Vessel"	4
4		Создание объекта control класса ControlBoard и его инициализация результатом вызова метода find с параметром "ControlBoard"	5
5		Вызов у объекта control метода setState с параметром 1	6
6		Объявление и инициализация нулем целочисленной переменной i	7

№	Предикат	Действия	№ перехода
7	i < поле m объекта area	Вызов у объекта control метода emitSignal с параметрами: сигналом order класса ControlBoard, строкой "OBSERVE"	8
			10
8		Вызов у объекта control метода emitSignal с параметрами: сигналом order класса ControlBoard, строкой "DRIFT"	9
9		Увеличение i на 1	7
10		Вызов у объекта control метода emitSignal с параметрами: сигналом order класса ControlBoard, строкой "STARTING"	11
11		Объявление булевой переменной isReach и ее инициализация ложным значением	12
12	isReach ложно	Вызов у объекта control метода emitSignal с параметрами: сигналом order класса ControlBoard, строкой "EXPLORE"	13
			15
13		Вызов у объекта control метода emitSignal с параметрами: сигналом order класса ControlBoard, строкой "FORWARD"	14
14	поле x поля cur объекта vessel == полю n объекта area	isReach присвоить истинное значение	12
			12
15		Вызов у объекта vessel метода emitSignal с параметрами: сигналом printFairway класса Vessel, строкой "Fairway"	16
16		Вызов у текущего объекта метода emitSignal с параметрами: сигналом order класса System, строкой "READ"	17

№	Предикат	Действия	№ перехода
17		Вызов у текущего объекта метода emitSignal с параметрами: сигналом order класса System, строкой "Turn off the ATM"	∅

3.5 Алгоритм метода order класса System

Функционал: Метод сигнала для объектов системы.

Параметры: string& s - ссылка на передаваемое сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода order класса System

№	Предикат	Действия	№ перехода
1		Выдача сигнала-команды	∅

3.6 Алгоритм метода cmdProcesser класса System

Функционал: Метод обработчика команд пользователя.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода cmdProcesser класса System

№	Предикат	Действия	№ перехода
1	s == "SHOWTREE"	Вызов у текущего объекта метода printStates	∅
			∅

3.7 Алгоритм конструктора класса Reader

Функционал: Параметризированный конструктор.

Параметры: Base* ancestor - указатель на головной объект; string name - имя объекта.

Алгоритм конструктора представлен в таблице 8.

Таблица 8 – Алгоритм конструктора класса Reader

№	Предикат	Действия	№ перехода
1		Делегирование конструктору базового класса Base параметров ancestor и name	∅

3.8 Алгоритм метода readln класса Reader

Функционал: Обработчик команд системы.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода readln класса Reader

№	Предикат	Действия	№ перехода
1	s != "READ"		∅
		Объявление строковой переменной info	2
2		Ввод с клавиатуры строки в переменную info	3
3		Вызов у текущего объекта метода emitSignal с параметрами сигнала read класса Reader и info	∅

3.9 Алгоритм метода read класса Reader

Функционал: Сигнал со считанной информацией.

Параметры: string& s - ссылка на передаваемое сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 10.

Таблица 10 – Алгоритм метода read класса Reader

№	Предикат	Действия	№ перехода
1		Выдача сигнала со считанными данными	∅

3.10 Алгоритм конструктора класса Display

Функционал: Параметризированный конструктор.

Параметры: Base* ancestor - указатель на головной объект; string name - имя объекта.

Алгоритм конструктора представлен в таблице 11.

Таблица 11 – Алгоритм конструктора класса Display

№	Предикат	Действия	№ перехода
1		Делегирование конструктору базового класса Base параметров ancestor и name	∅

3.11 Алгоритм метода print класса Display

Функционал: Обработчик запросов о выводе данных.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 12.

Таблица 12 – Алгоритм метода print класса Display

№	Предикат	Действия	№ перехода
1	s == "READ"		∅

№	Предикат	Действия	№ перехода
		Вывод на экран значения параметра s и перевода строки	∅

3.12 Алгоритм конструктора класса Aquatory

Функционал: Параметризированный конструктор.

Параметры: Base* ancestor - указатель на головной объект; string name - имя объекта.

Алгоритм конструктора представлен в таблице 13.

Таблица 13 – Алгоритм конструктора класса Aquatory

№	Предикат	Действия	№ перехода
1		Делегирование конструктору базового класса Base параметров ancestor и name	2
2		Инициализация поля curLine нулевым значением	∅

3.13 Алгоритм метода setProperties класса Aquatory

Функционал: Обработчик для установки параметров.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 14.

Таблица 14 – Алгоритм метода setProperties класса Aquatory

№	Предикат	Действия	№ перехода
1		Объявление переменной строкового потока ss	2
2		Загрузка в ss значения строки s	3
3		Объявление строковой переменной val	4
4		Считывание из потока ss значения в переменную	5

№	Предикат	Действия	№ перехода
		val	
5		Полю n присваивается преобразованное в число значение строки val	6
6		Считывание из потока ss значения в переменную val	7
7		Полю m присваивается преобразованное в число значение строки val	8
8		Инициализация поля area новой областью данных размера n указателей на целочисленные области данных	9
9		Объявление и инициализация нулем целочисленной переменной i	10
10	i < поле n	Элементу поля area с индексом i присваивается значение новой области целочисленных данных размера m	11
			∅
11		Увеличение i на 1	10

3.14 Алгоритм метода getSignal класса Aquatory

Функционал: Обработчик для установки глубин.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 15.

Таблица 15 – Алгоритм метода getSignal класса Aquatory

№	Предикат	Действия	№ перехода
1		Объявление переменной строкового потока ss	2
2		Загрузка в ss значения строки s	3

№	Предикат	Действия	№ перехода
3		Объявление строковой переменной val	4
4		Объявление целочисленных переменных xl и yl	5
5		Считывание из потока ss значения в переменную val	6
6		Переменной xl присваивается преобразованное в число значение строки val	7
7		Считывание из потока ss значения в переменную val	8
8		Переменной yl присваивается преобразованное в число значение строки val	9
9	xl >= 1 && xl <= поле n && yl >= 1 && yl <= поле m	Вызов у текущего объекта метода emitSignal с параметрами сигнала returnDepth класса Aquatory и преобразованного в строку значения поля area по индексам xl-1 и yl-1	∅
			∅

3.15 Алгоритм метода returnDepth класса Aquatory

Функционал: Сигнал локатору.

Параметры: string& s - ссылка на передаваемое сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 16.

Таблица 16 – Алгоритм метода returnDepth класса Aquatory

№	Предикат	Действия	№ перехода
1		Выдача сигнала с глубиной клетки	∅

3.16 Алгоритм метода setLine класса Aquatory

Функционал: Обработчик сигналов локатора.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 17.

Таблица 17 – Алгоритм метода setLine класса Aquatory

№	Предикат	Действия	№ перехода
1		Объявление переменной строкового потока ss	2
2		Загрузка в ss значения строки s	3
3		Объявление строковой переменной val	4
4		Объявление и инициализация нулем целочисленной переменной j	5
5	j < поле m	Считывание из потока ss значения в переменную val	6
			8
6		Полю area с индексами curLine и j присвоение преобразованного в число значения строки val	7
7		Увеличение j на 1	5
8		Увеличение поля curLine на один	∅

3.17 Алгоритм конструктора класса Vessel

Функционал: Параметризированный конструктор.

Параметры: Base* ancestor - указатель на головной объект; string name - имя объекта.

Алгоритм конструктора представлен в таблице 18.

Таблица 18 – Алгоритм конструктора класса Vessel

№	Предикат	Действия	№ перехода
1		Делегирование конструктору базового класса Base параметров ancestor и name	2
2		Инициализация поля cur значением нового указателя на структуру Cell с параметрами 1 и 1	3
3		Инициализация поля course значением EAST перечисляемого типа Direction	4
4		Инициализация поля entrance значением нового указателя на структуру Cell с параметрами 2 и -1	∅

3.18 Алгоритм метода printFairway класса Vessel

Функционал: Сигнал для вывода фарватера.

Параметры: string& s - ссылка на передаваемое сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 19.

Таблица 19 – Алгоритм метода printFairway класса Vessel

№	Предикат	Действия	№ перехода
1		Ссылке s добавление конкатенации строк " (1, " , преобразованного в строку значения у поля entrance, ")", " (2, " , преобразованного в строку значения у поля entrance, ")" и поля fairway	∅

3.19 Алгоритм метода setProperties класса Vessel

Функционал: Обработчик для установки параметров.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 20.

Таблица 20 – Алгоритм метода setProperties класса Vessel

№	Предикат	Действия	№ перехода
1		Объявление переменной строкового потока ss	2
2		Загрузка в ss значения строки s	3
3		Объявление строковой переменной val	4
4		Считывание из потока ss значения в переменную val	5
5		Полю minDepth присваивается преобразованное в число значение строки val	∅

3.20 Алгоритм метода move класса Vessel

Функционал: Обработчик команды пульта управления.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 21.

Таблица 21 – Алгоритм метода move класса Vessel

№	Предикат	Действия	№ перехода
1	s == "DRIФT"	Увеличение поля y поля cur на 1	∅
			2
2	s == "ФORWARD"		3
			∅
3	поле course == EAST	Увеличение поля y поля cur на 1	8
			4
4	поле course == SEAST	Увеличение поля y поля cur на 1, x на 1	8
			5
5	поле course == SOUTH	Увеличение поля x поля cur на 1	8
			6
6	поле course == SWEST	Уменьшение поля y поля cur на 1, увеличение поля x на 1	8

№	Предикат	Действия	№ перехода
			7
7	поле course == WEST	Уменьшение поля y поля cur на 1	8
			8
8		Полю fairway добавление конкатенации строк " (", преобразованного в строку поля x, ", ", преобразованного в строку поля y, ")"	∅

3.21 Алгоритм конструктора класса ControlBoard

Функционал: Параметризированный конструктор.

Параметры: Base* ancestor - указатель на головной объект; string name - имя объекта.

Алгоритм конструктора представлен в таблице 22.

Таблица 22 – Алгоритм конструктора класса ControlBoard

№	Предикат	Действия	№ перехода
1		Делегирование конструктору базового класса Base параметров ancestor и name	2
2		Инициализация поля maxDepth нулем	3
3		Инициализация поля entranceFound ложным значением	∅

3.22 Алгоритм метода order класса ControlBoard

Функционал: Сигнал-команда.

Параметры: string& s - ссылка на передаваемое сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 23.

Таблица 23 – Алгоритм метода order класса ControlBoard

№	Предикат	Действия	№ перехода
1	s == "STARTING"	Полю cur головного объекта ancestor класса Vessel присваивается значение поля entrance того же объекта	2
			3
2		Полю entranceFound присваивается истинное значение	∅
3	s == "EXPLORE"	Полю maxDepth присваивается нулевое значение	∅
			∅

3.23 Алгоритм метода getDepth класса ControlBoard

Функционал: Обработчик локатора.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 24.

Таблица 24 – Алгоритм метода getDepth класса ControlBoard

№	Предикат	Действия	№ перехода
1		Объявление переменной строкового потока ss	2
2		Загрузка в ss значения строки s	3
3		Объявление строковой переменной val	4
4		Считывание из потока ss значения в переменную val	5
5		Объявление целочисленной переменной depth и ее инициализация преобразованной в число строкой val	6
6		Считывание из потока ss значения в переменную val	7

№	Предикат	Действия	№ перехода
7		Объявление переменной dir перечисляемого типа Direction и ее инициализация преобразованной строкой val	8
8		Объявление целочисленной переменной minDepth и ее инициализация полем minDepth головного объекта ancestor класса Vessel	9
9	depth >= minDepth && depth > поле maxDepth	Полю maxDepth присвоение значения depth	10
			∅
10	поле entranceFound ложно	Полю у поля entrance головного объекта ancestor класса Vessel присвоение поля у поля cur того же объекта	∅
		Полю course головного объекта ancestor класса Vessel присвоение значения dir	∅

3.24 Алгоритм конструктора класса Locator

Функционал: Параметризированный конструктор.

Параметры: Base* ancestor - указатель на головной объект; string name - имя объекта.

Алгоритм конструктора представлен в таблице 25.

Таблица 25 – Алгоритм конструктора класса Locator

№	Предикат	Действия	№ перехода
1		Делегирование конструктору базового класса Base параметров ancestor и name	2
2		Инициализация поля curDir значением EAST перечисляемого типа Direction	∅

3.25 Алгоритм метода locate класса Locator

Функционал: Сигнал локатора.

Параметры: string& s - ссылка на передаваемое сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 26.

Таблица 26 – Алгоритм метода locate класса Locator

№	Предикат	Действия	№ перехода
1		Выдача сигнала с координатой запрашиваемой клетки	∅

3.26 Алгоритм метода returnDepth класса Locator

Функционал: Сигнал возврата исследованной глубины.

Параметры: string& s - ссылка на передаваемое сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 27.

Таблица 27 – Алгоритм метода returnDepth класса Locator

№	Предикат	Действия	№ перехода
1		Ссылке s присоединение " " и преобразованного в строку поля curDir	∅

3.27 Алгоритм метода receiveCmd класса Locator

Функционал: Обработчик команд пульта управления.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 28.

Таблица 28 – Алгоритм метода receiveCmd класса Locator

№	Предикат	Действия	№ перехода
1	s == "OBSERVE"	Объявление переменной curLoc типа Cell и ее инициализация значением поля cur головного объекта ancestor класса Vessel	2
			4
2		Полю curDir присвоение SOUTH	3
3		Вызов у текущего объекта метода emitSignal с параметрами сигнала locate класса Locator и преобразованными в строку значениями полей x (увеличенного на 1) и y переменной curLoc	∅
4	s == "EXPLORE"	Объявление и инициализация целочисленной переменной x значением поля x поля cur головного объекта ancestor класса Vessel	5
			∅
5		Объявление и инициализация целочисленной переменной y значением поля y поля cur головного объекта ancestor класса Vessel	6
6		Полю curDir присвоение SOUTH	7
7		Вызов у текущего объекта метода emitSignal с параметрами сигнала locate класса Locator и преобразованными в строку значениями x+1 и y	8
8		Полю curDir присвоение SEAST	9
9		Вызов у текущего объекта метода emitSignal с параметрами сигнала locate класса Locator и преобразованными в строку значениями x+1 и y+1	10
10		Полю curDir присвоение SWEST	11
11		Вызов у текущего объекта метода emitSignal с параметрами сигнала locate класса Locator и преобразованными в строку значениями x+1 и y-1	12

№	Предикат	Действия	№ перехода
12		Полю curDir присвоение EAST	13
13		Вызов у текущего объекта метода emitSignal с параметрами сигнала locate класса Locator и преобразованными в строку значениями x и y+1	14
14		Полю curDir присвоение WEST	15
15		Вызов у текущего объекта метода emitSignal с параметрами сигнала locate класса Locator и преобразованными в строку значениями x и y-1	∅

3.28 Алгоритм метода receiveDepth класса Locator

Функционал: Обработчик ответа от акватории.

Параметры: string s - полученное сообщение.

Возвращаемое значение: Нет.

Алгоритм метода представлен в таблице 29.

Таблица 29 – Алгоритм метода receiveDepth класса Locator

№	Предикат	Действия	№ перехода
1		Вызов у текущего объекта метода emitSignal с параметрами сигнала returnDepth класса Locator и строкой s	∅

3.29 Алгоритм функции main

Функционал: Запускает работу системы.

Параметры: нет.

Возвращаемое значение: Целочисленное значение.

Алгоритм функции представлен в таблице 30.

Таблица 30 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1		Создание объекта system класса System с параметром конструктора nullptr	2
2		Вызов метода buildTree у объекта system	3
3		Возврат результата вызова метода run у объекта system	∅

3.30 Алгоритм деструктора класса System

Функционал: Деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 31.

Таблица 31 – Алгоритм деструктора класса System

№	Предикат	Действия	№ перехода
1		Удаление текущего объекта	∅

3.31 Алгоритм деструктора класса Reader

Функционал: Деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 32.

Таблица 32 – Алгоритм деструктора класса Reader

№	Предикат	Действия	№ перехода
1		Удаление текущего объекта	∅

3.32 Алгоритм деструктора класса Display

Функционал: Деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 33.

Таблица 33 – Алгоритм деструктора класса Display

№	Предикат	Действия	№ перехода
1		Удаление текущего объекта	∅

3.33 Алгоритм деструктора класса Aquatory

Функционал: Деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 34.

Таблица 34 – Алгоритм деструктора класса Aquatory

№	Предикат	Действия	№ перехода
1		Объявление и инициализация целочисленной переменной i нулем	2
2	i < поле n	Удаление данных по указателю поля area по индексу i	3
		Удаление данных по указателю поля area	∅
3		Увеличение i на 1	2

3.34 Алгоритм деструктора класса Vessel

Функционал: Деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 35.

Таблица 35 – Алгоритм деструктора класса Vessel

№	Предикат	Действия	№ перехода
1		Удаление данных по адресу поля cur	2
2		Удаление данных по адресу поля entrance	∅

3.35 Алгоритм деструктора класса ControlBoard

Функционал: Деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 36.

Таблица 36 – Алгоритм деструктора класса ControlBoard

№	Предикат	Действия	№ перехода
1		Удаление текущего объекта	∅

3.36 Алгоритм деструктора класса Locator

Функционал: Деструктор.

Параметры: нет.

Алгоритм деструктора представлен в таблице 37.

Таблица 37 – Алгоритм деструктора класса Locator

№	Предикат	Действия	№ перехода
1		Удаление текущего объекта	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-26.

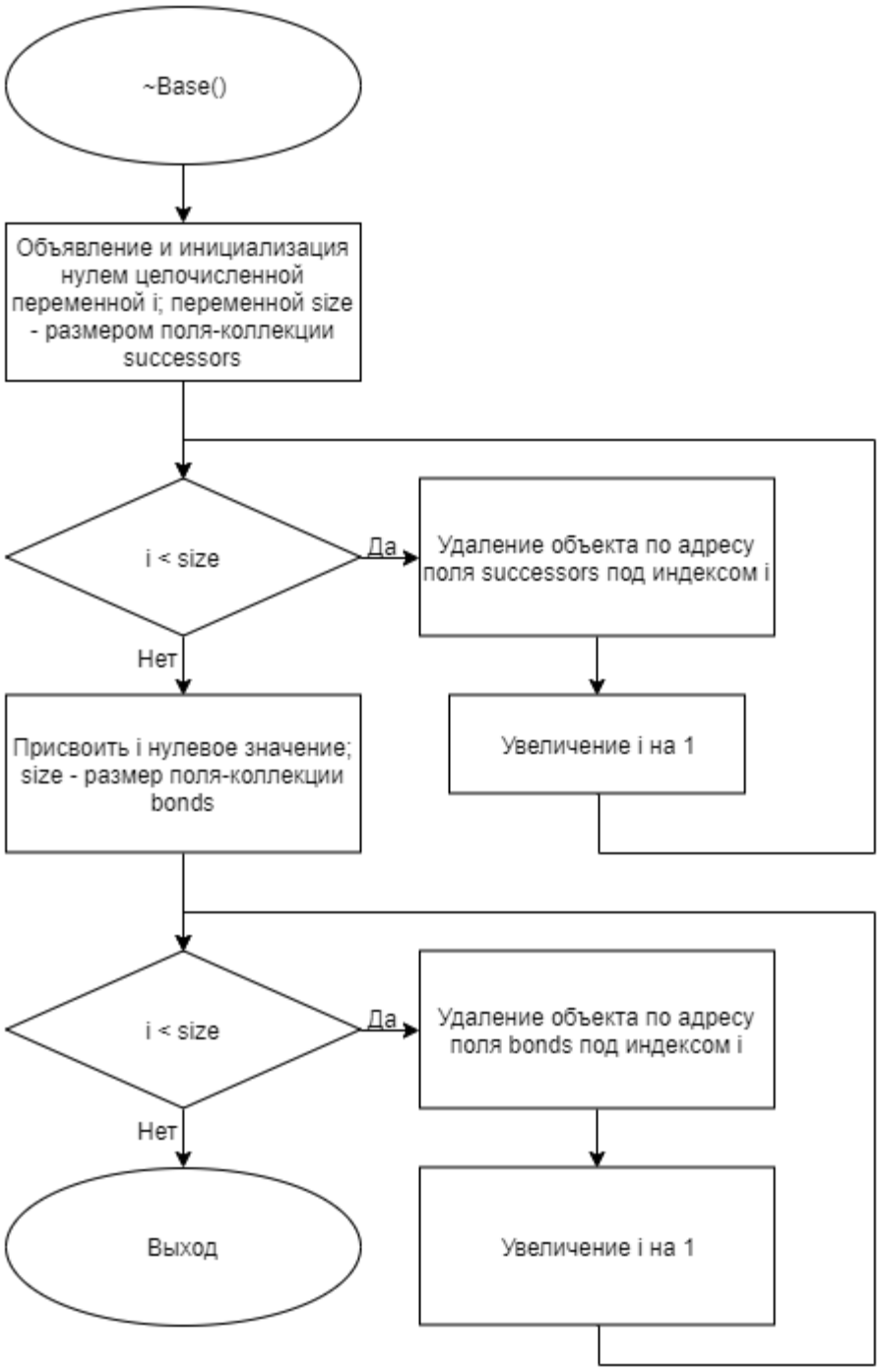

Рисунок 1 – Блок-схема алгоритма

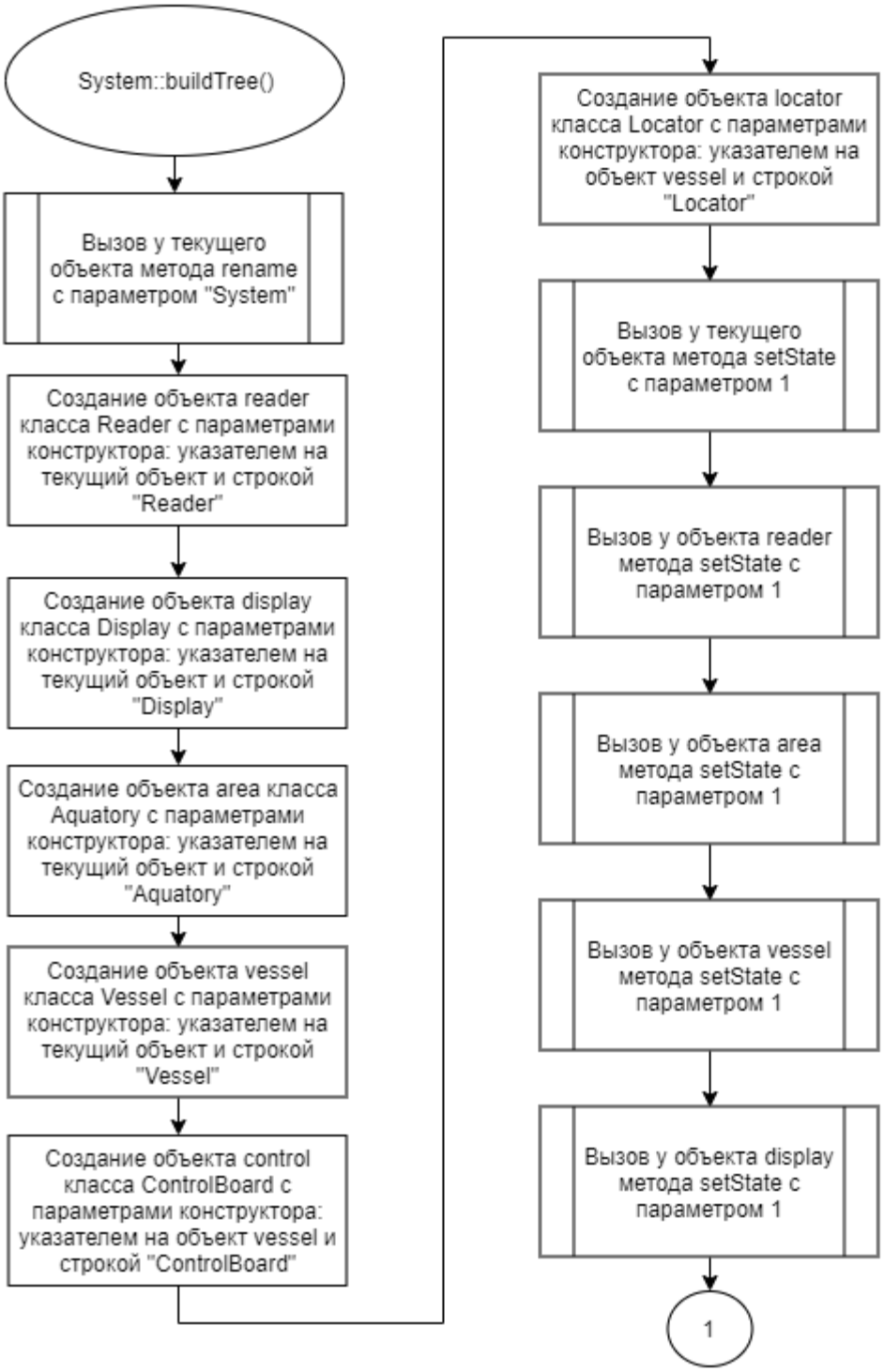

Рисунок 2 – Блок-схема алгоритма

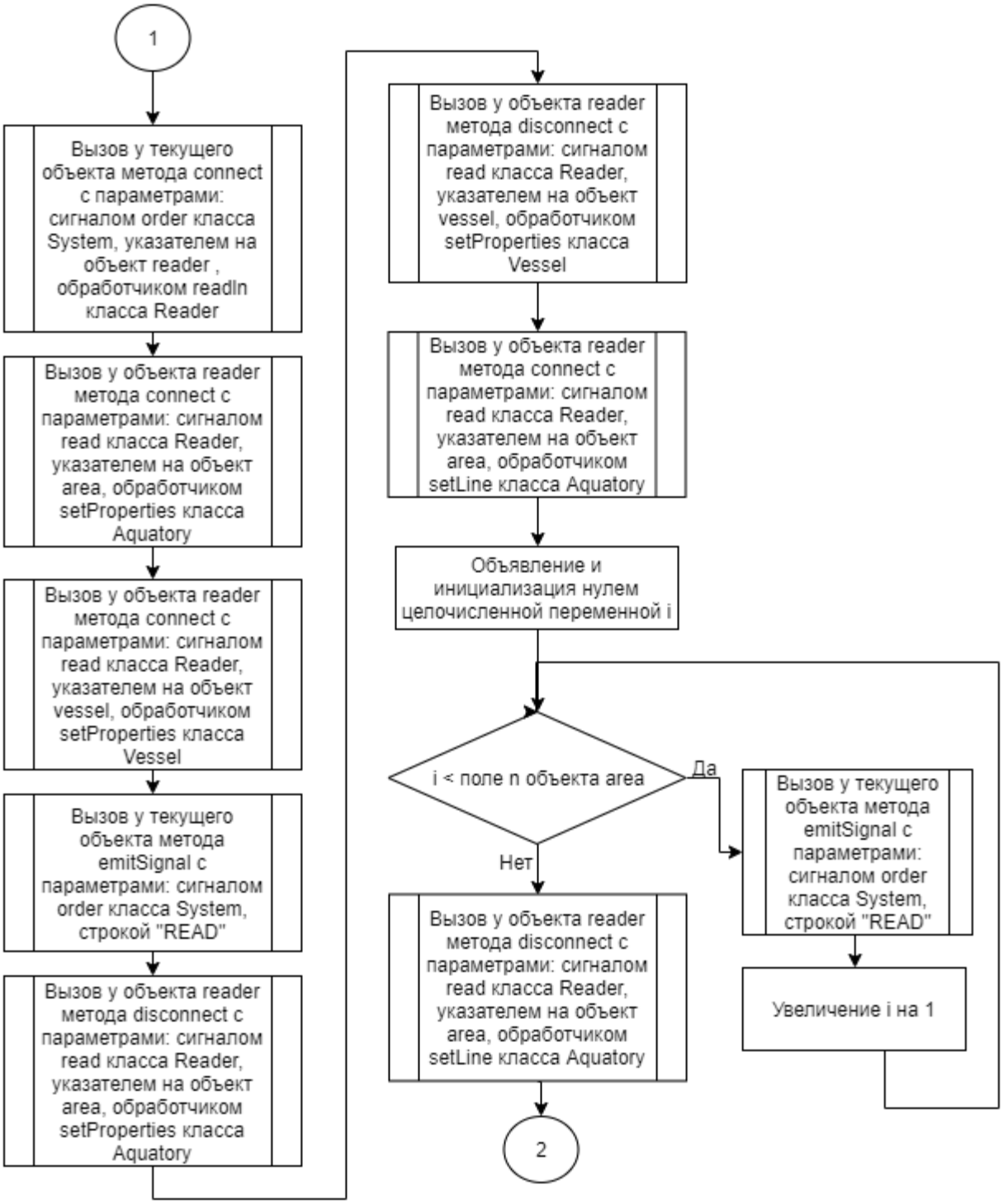

Рисунок 3 – Блок-схема алгоритма

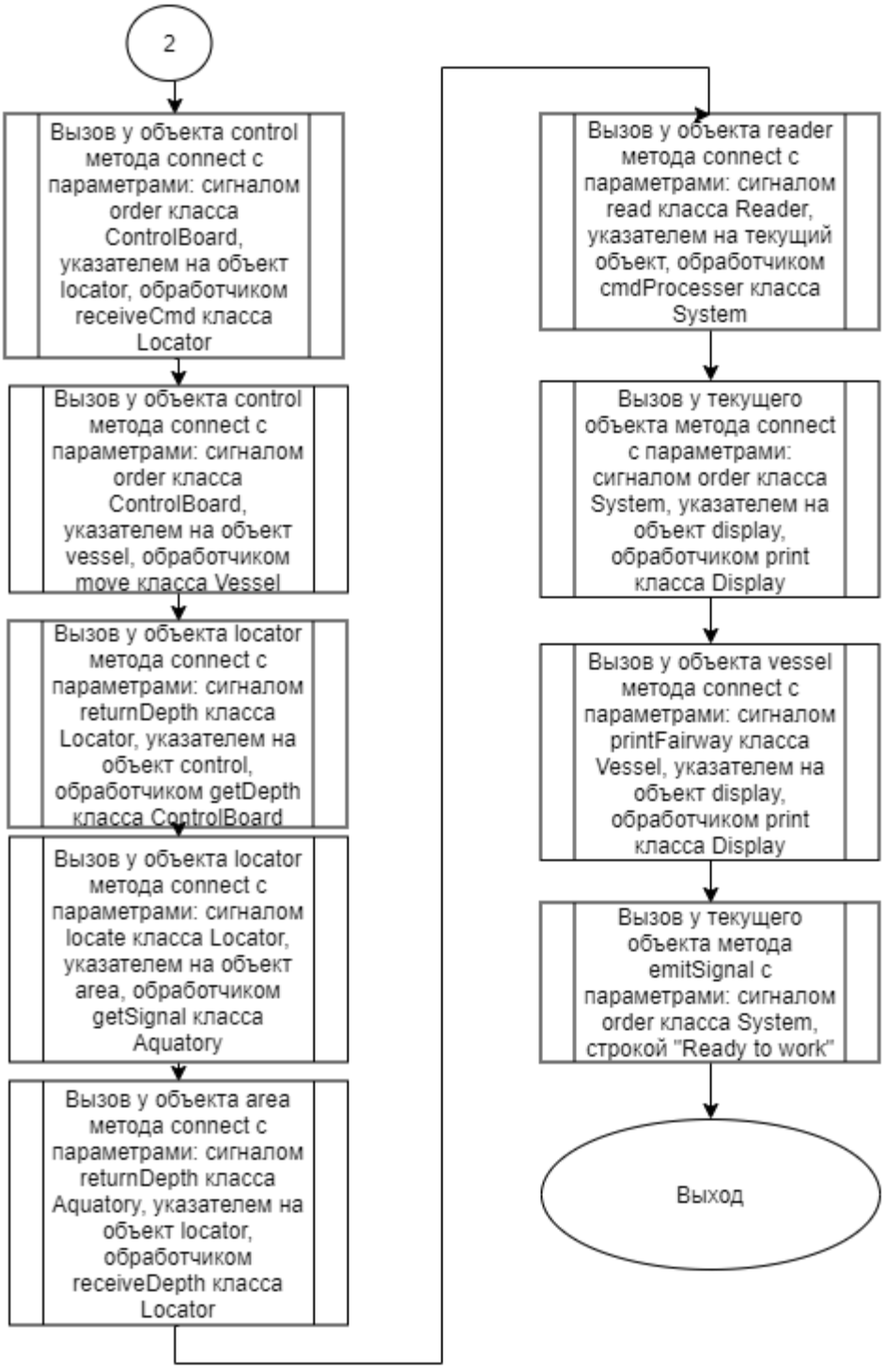

Рисунок 4 – Блок-схема алгоритма

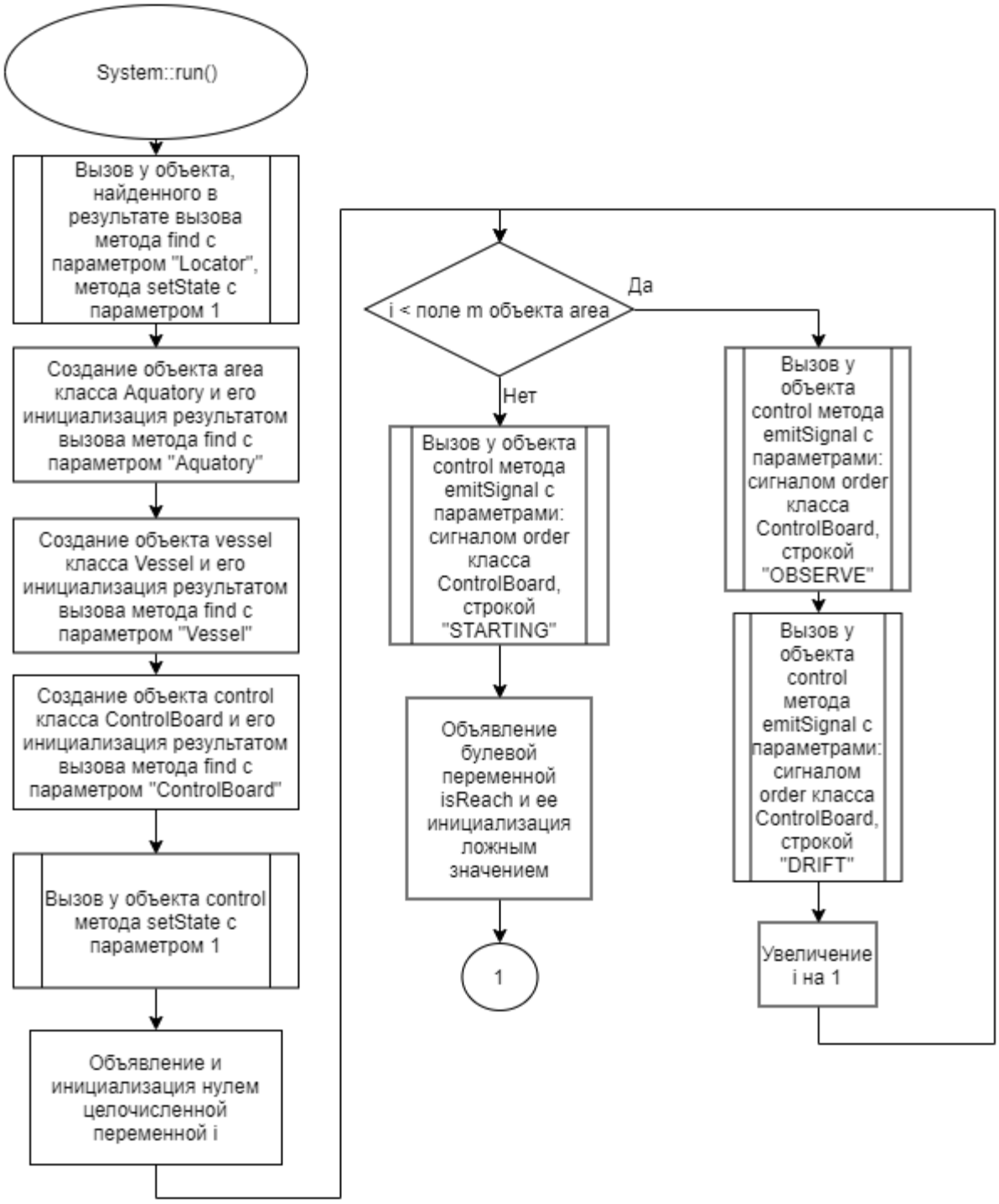

Рисунок 5 – Блок-схема алгоритма

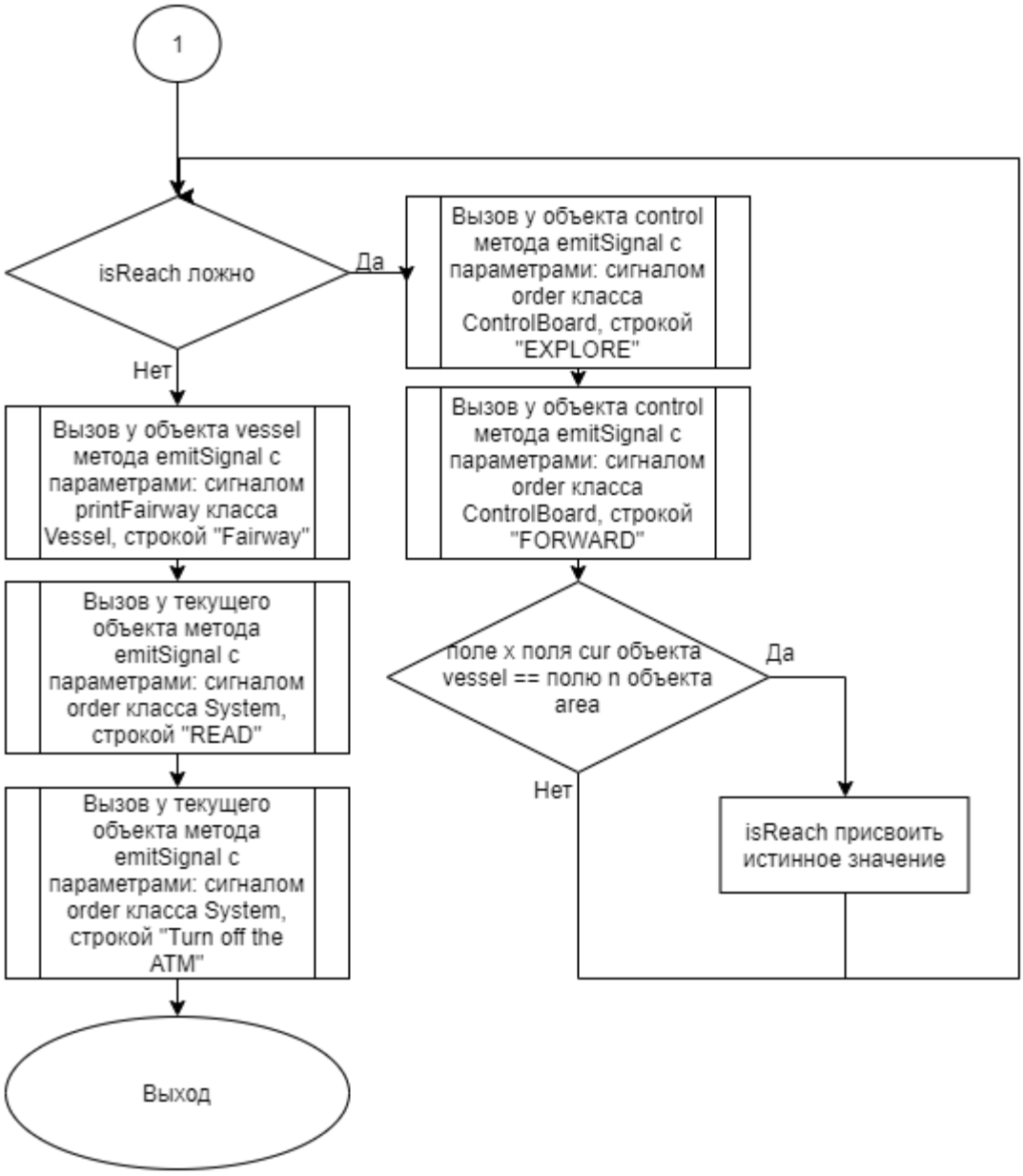

Рисунок 6 – Блок-схема алгоритма

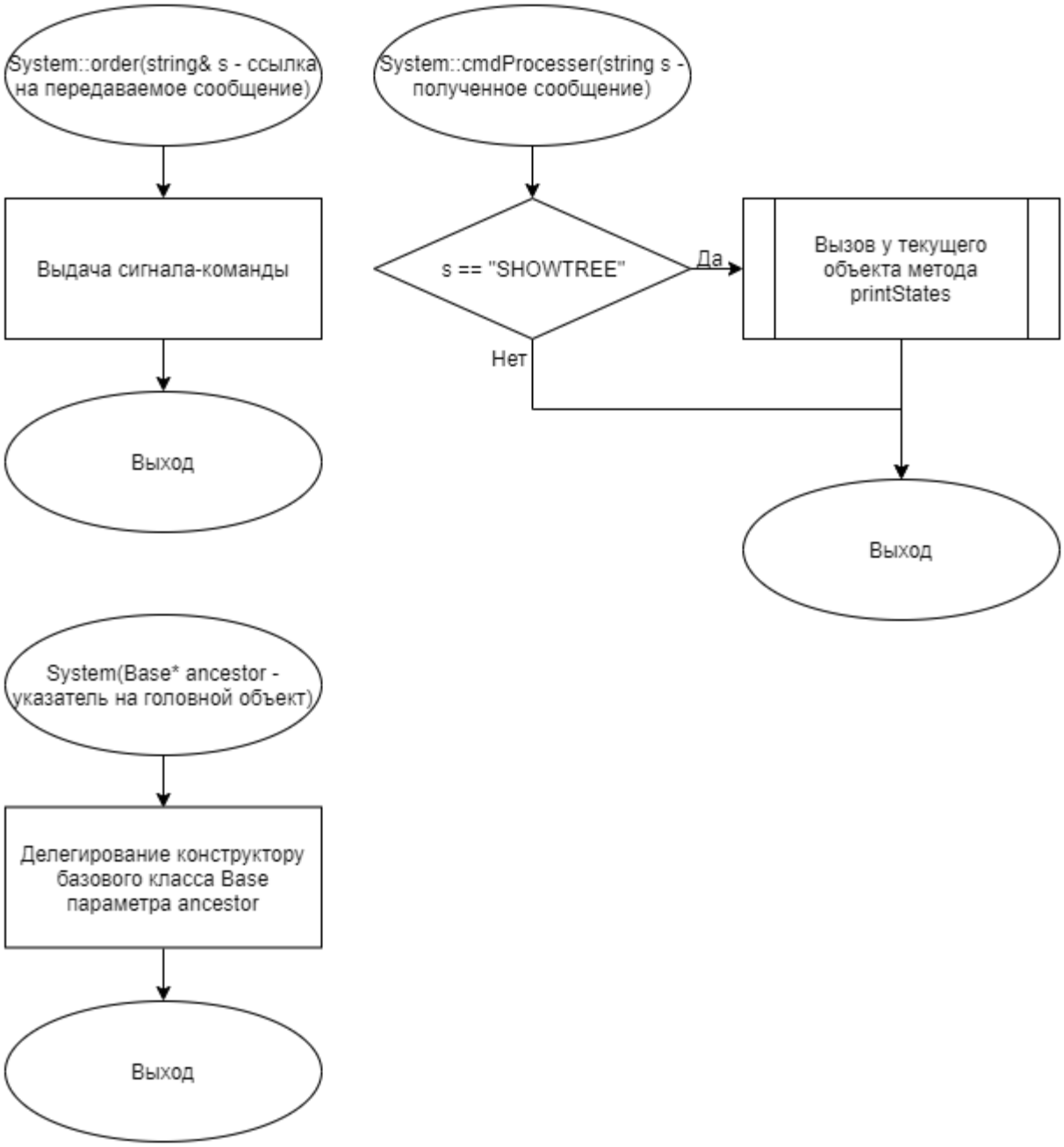

Рисунок 7 – Блок-схема алгоритма

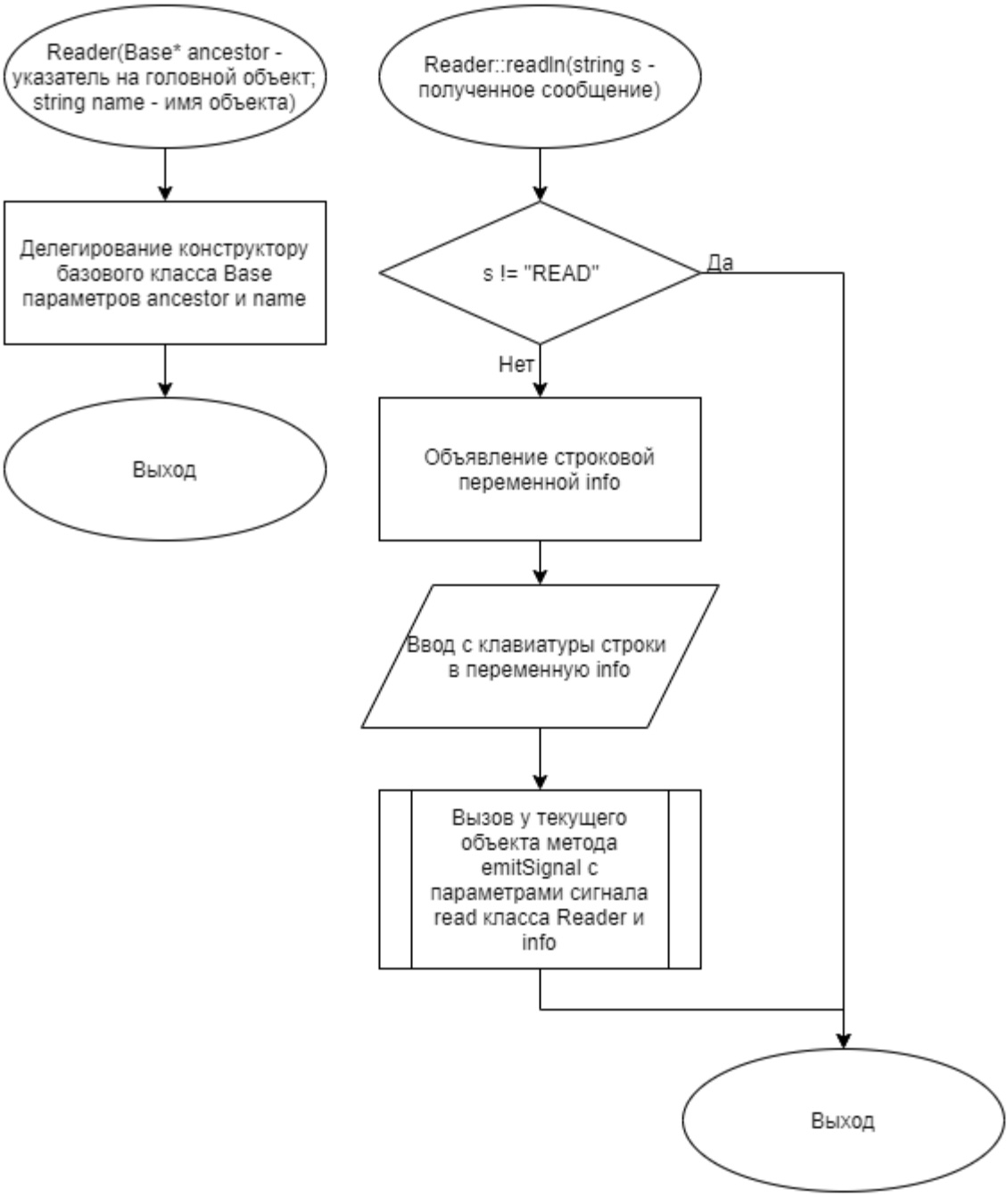

Рисунок 8 – Блок-схема алгоритма

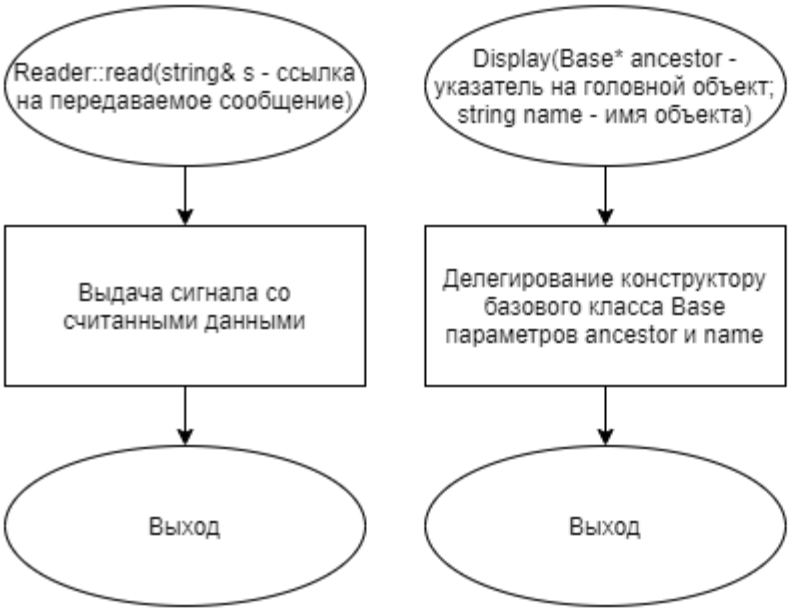

Рисунок 9 – Блок-схема алгоритма

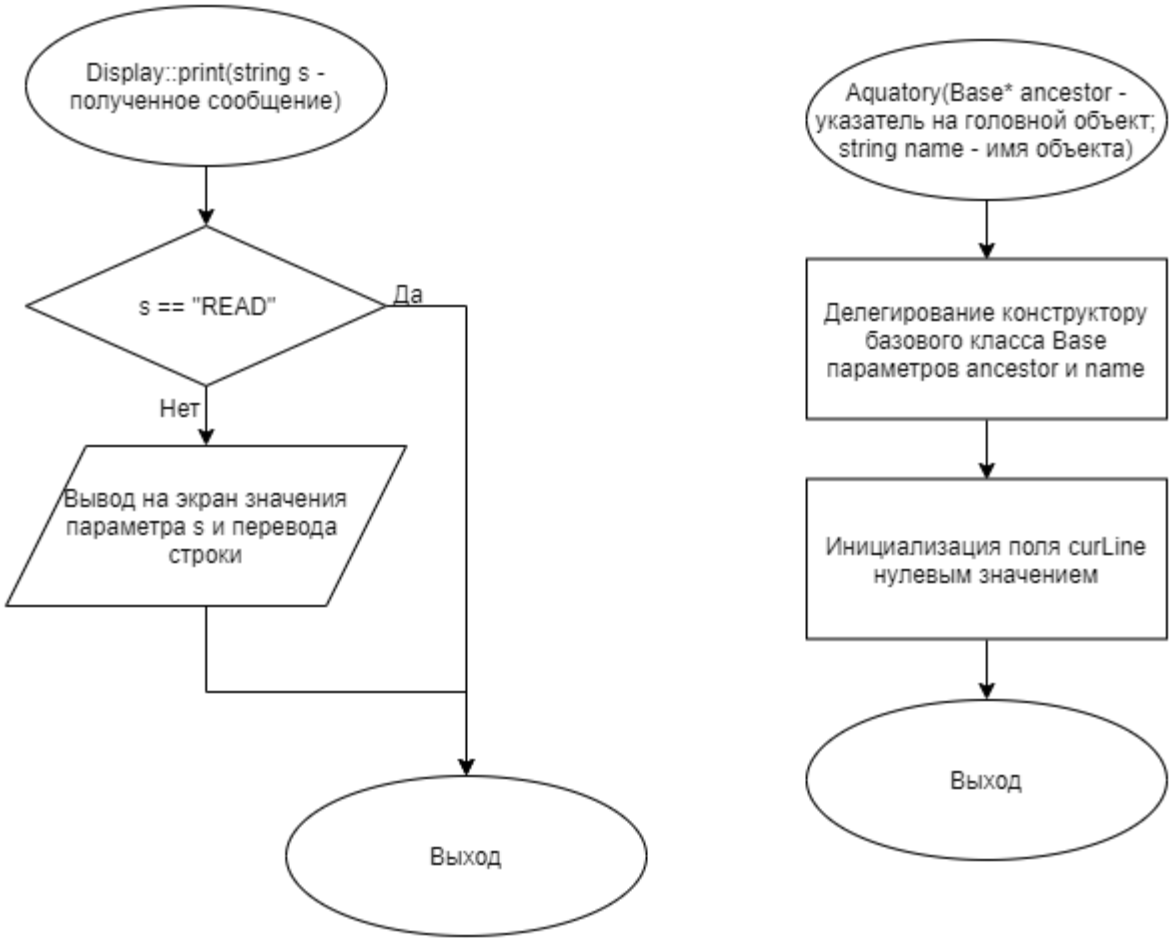

Рисунок 10 – Блок-схема алгоритма

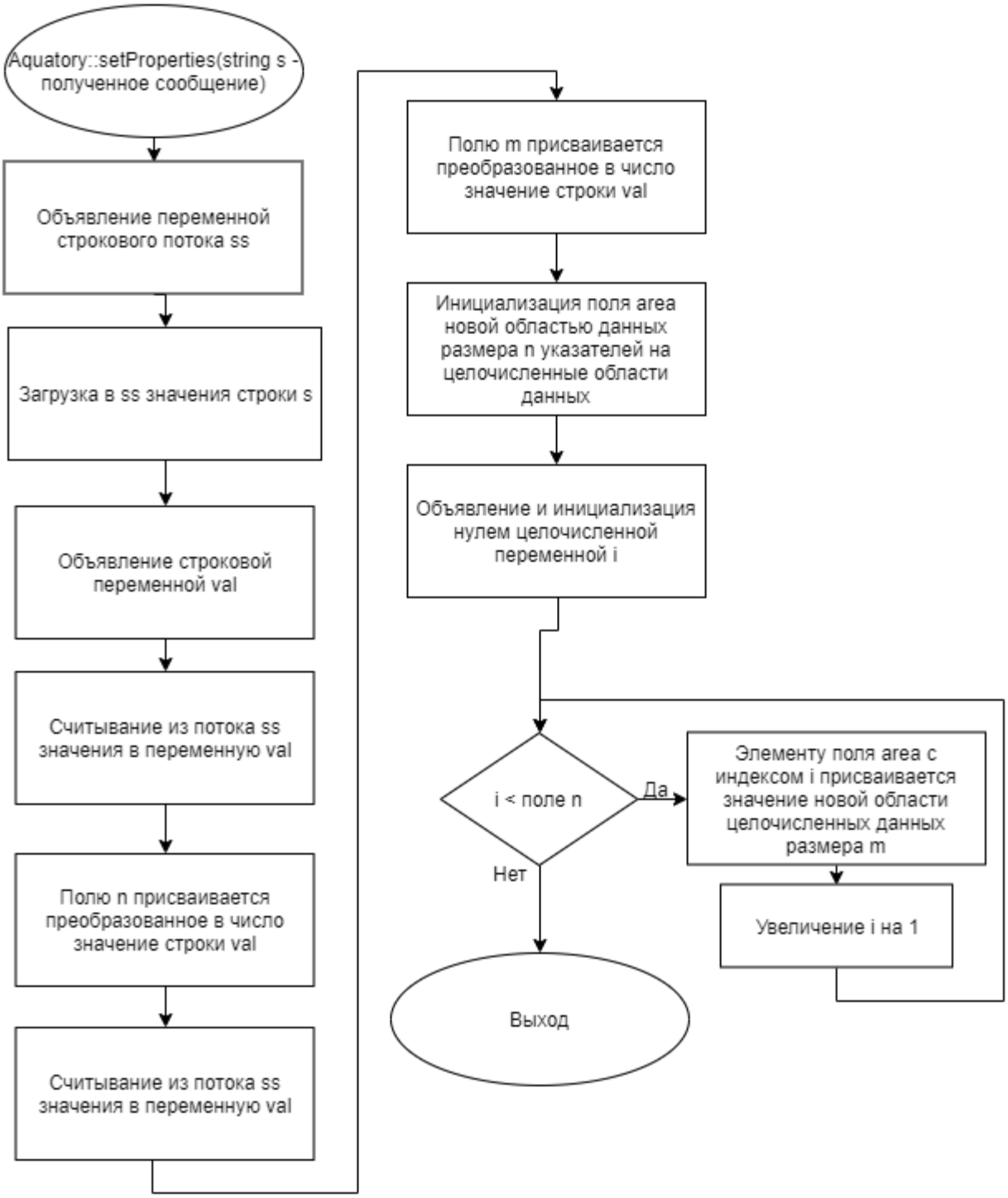

Рисунок 11 – Блок-схема алгоритма

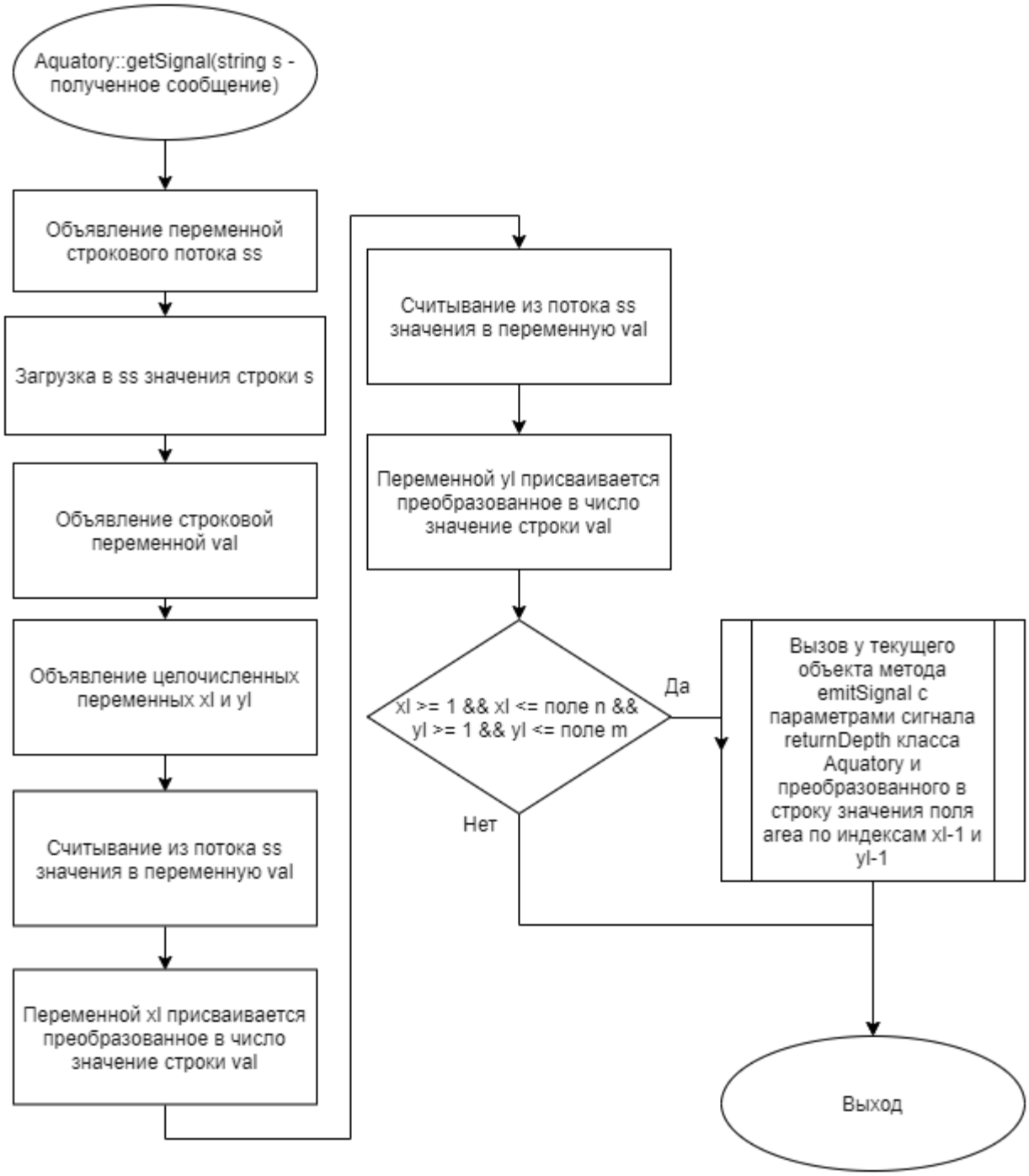

Рисунок 12 – Блок-схема алгоритма

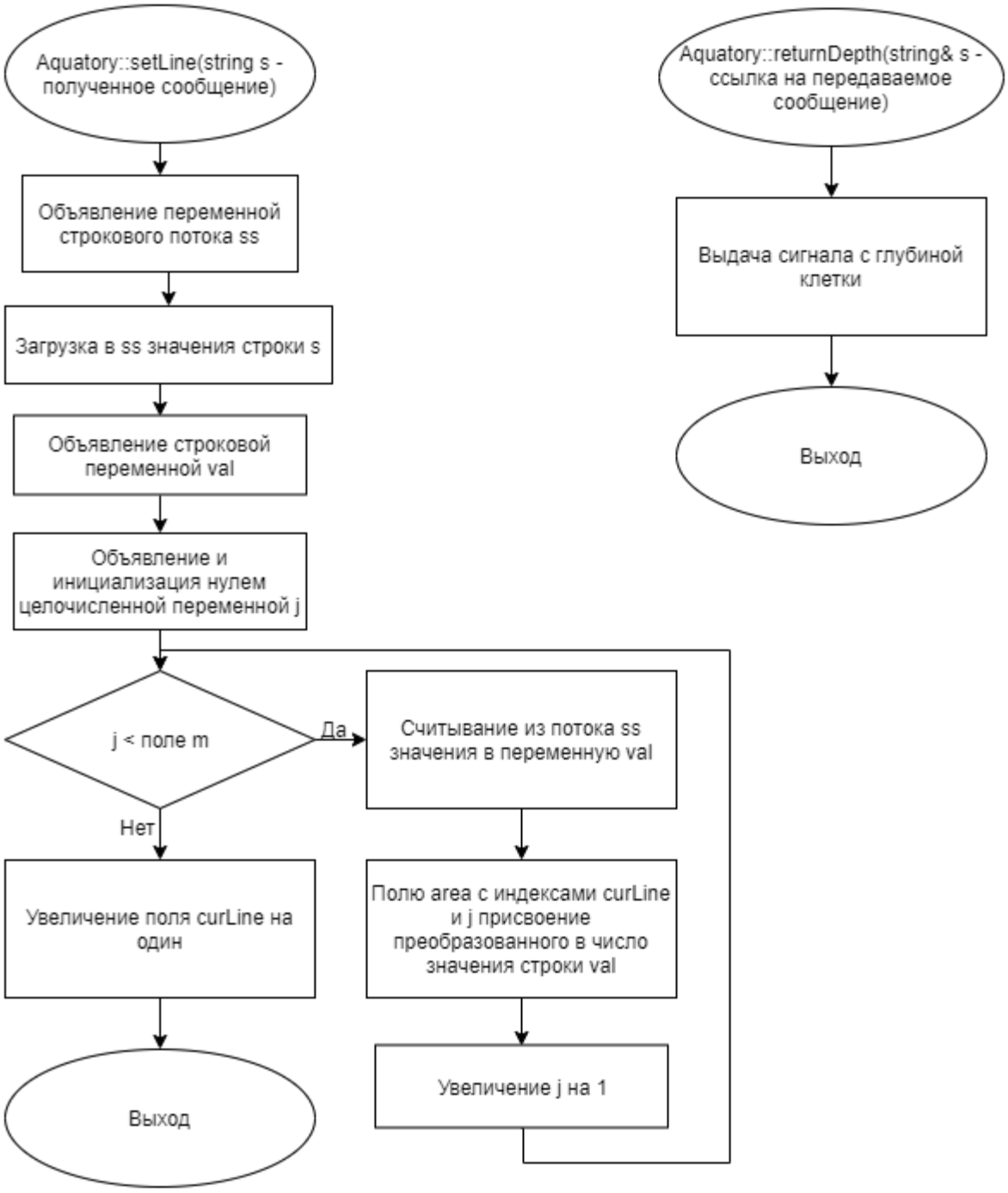

Рисунок 13 – Блок-схема алгоритма

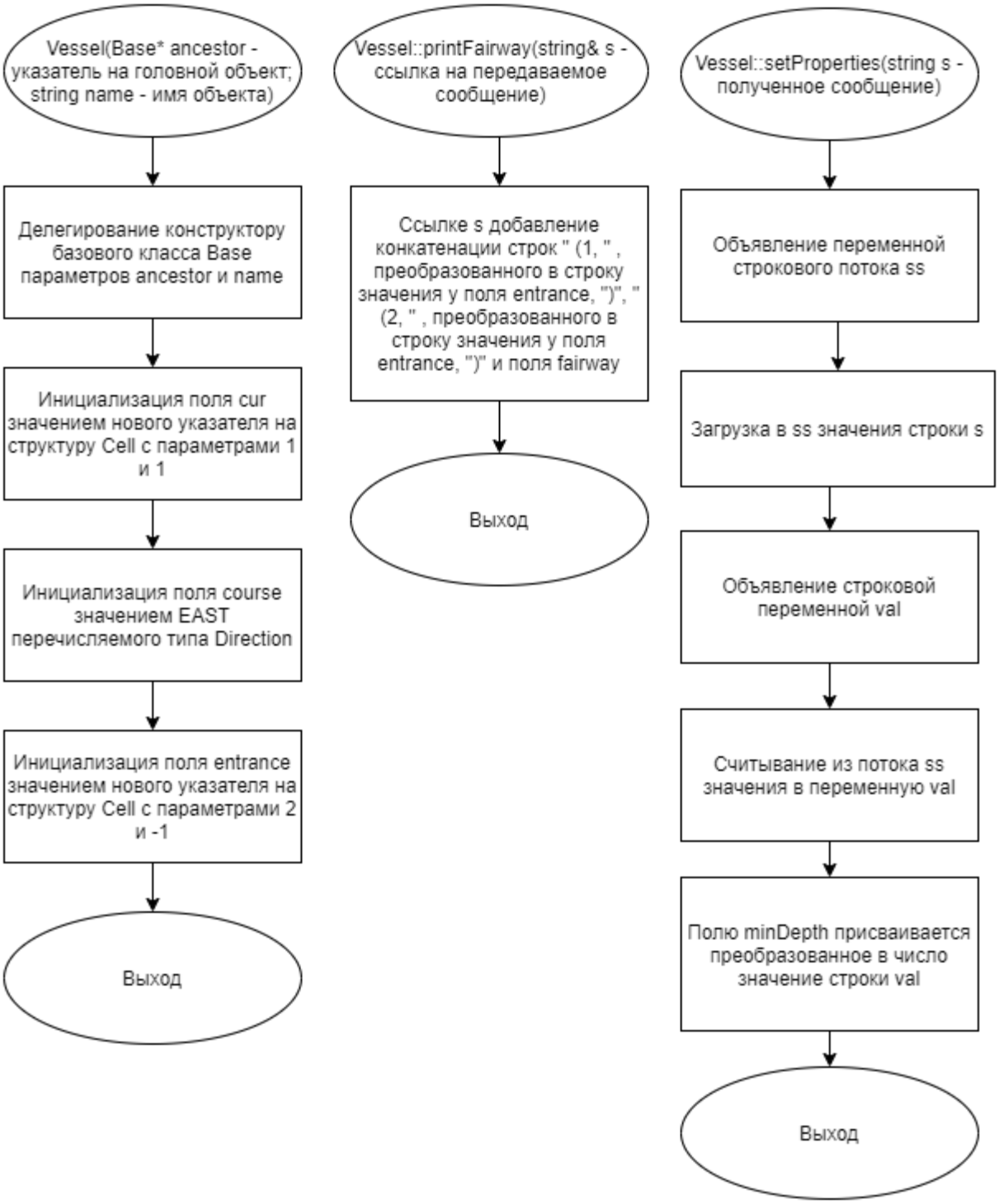

Рисунок 14 – Блок-схема алгоритма

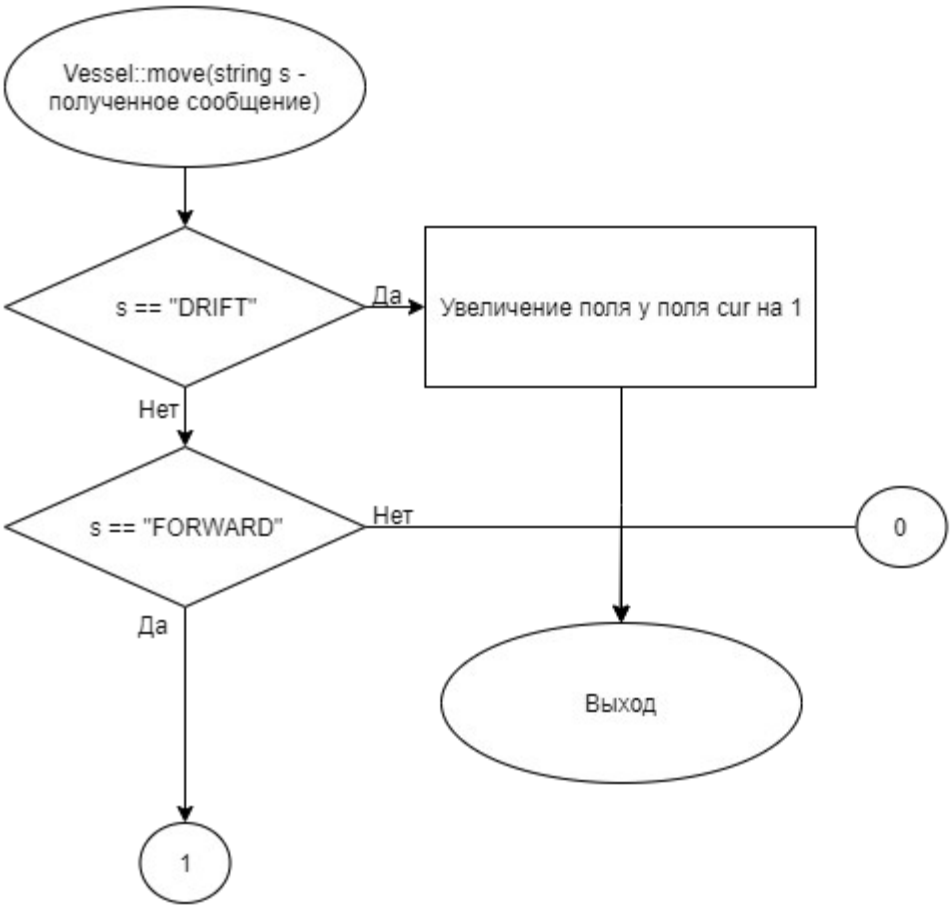

Рисунок 15 – Блок-схема алгоритма

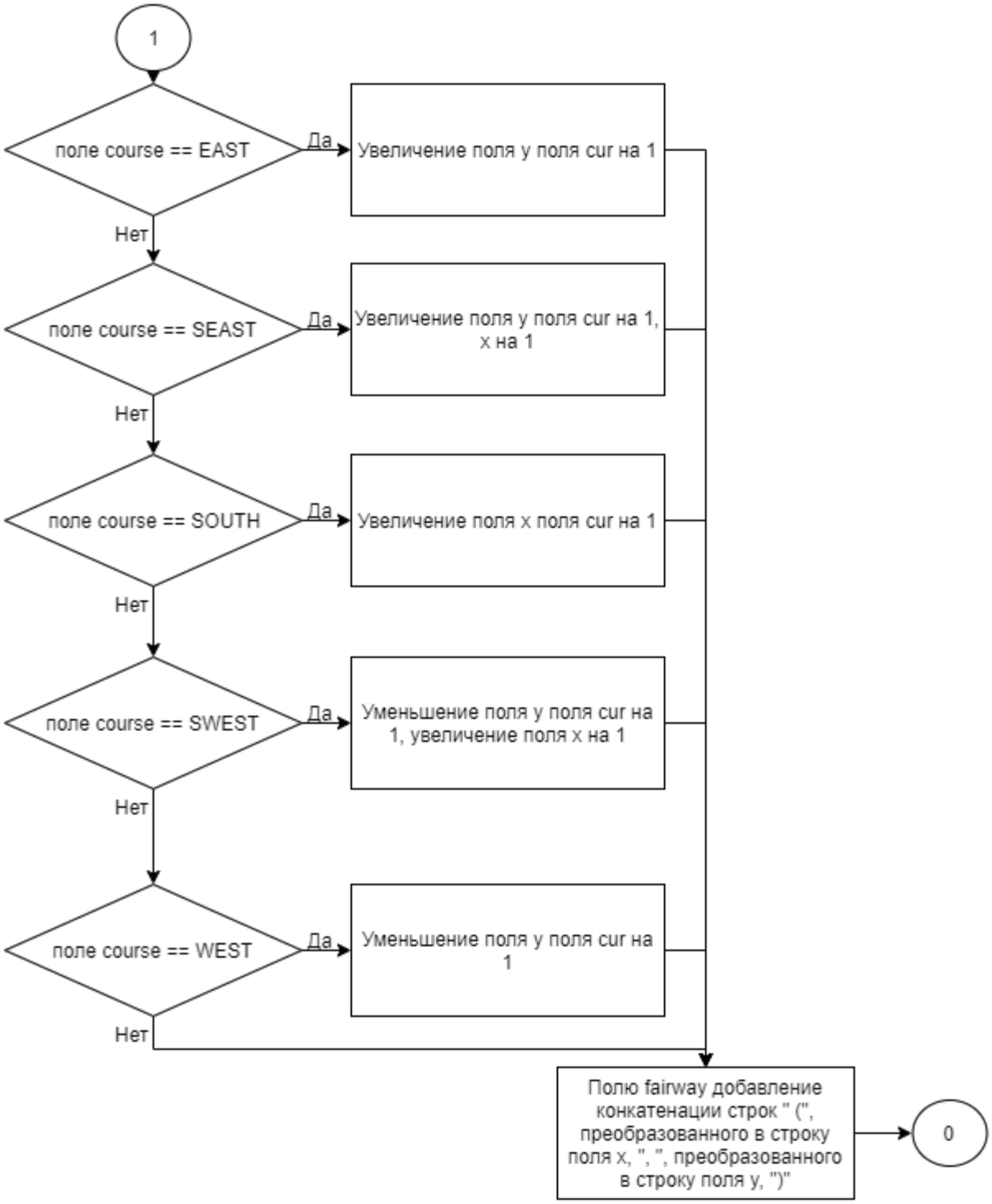

Рисунок 16 – Блок-схема алгоритма

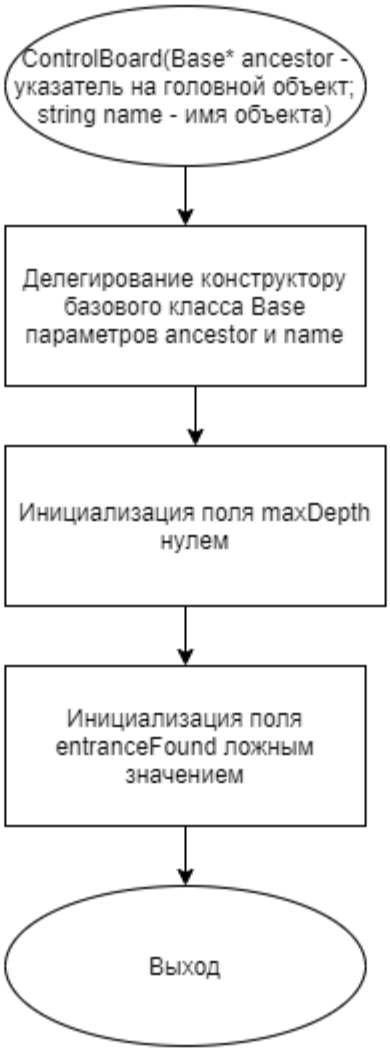

Рисунок 17 – Блок-схема алгоритма

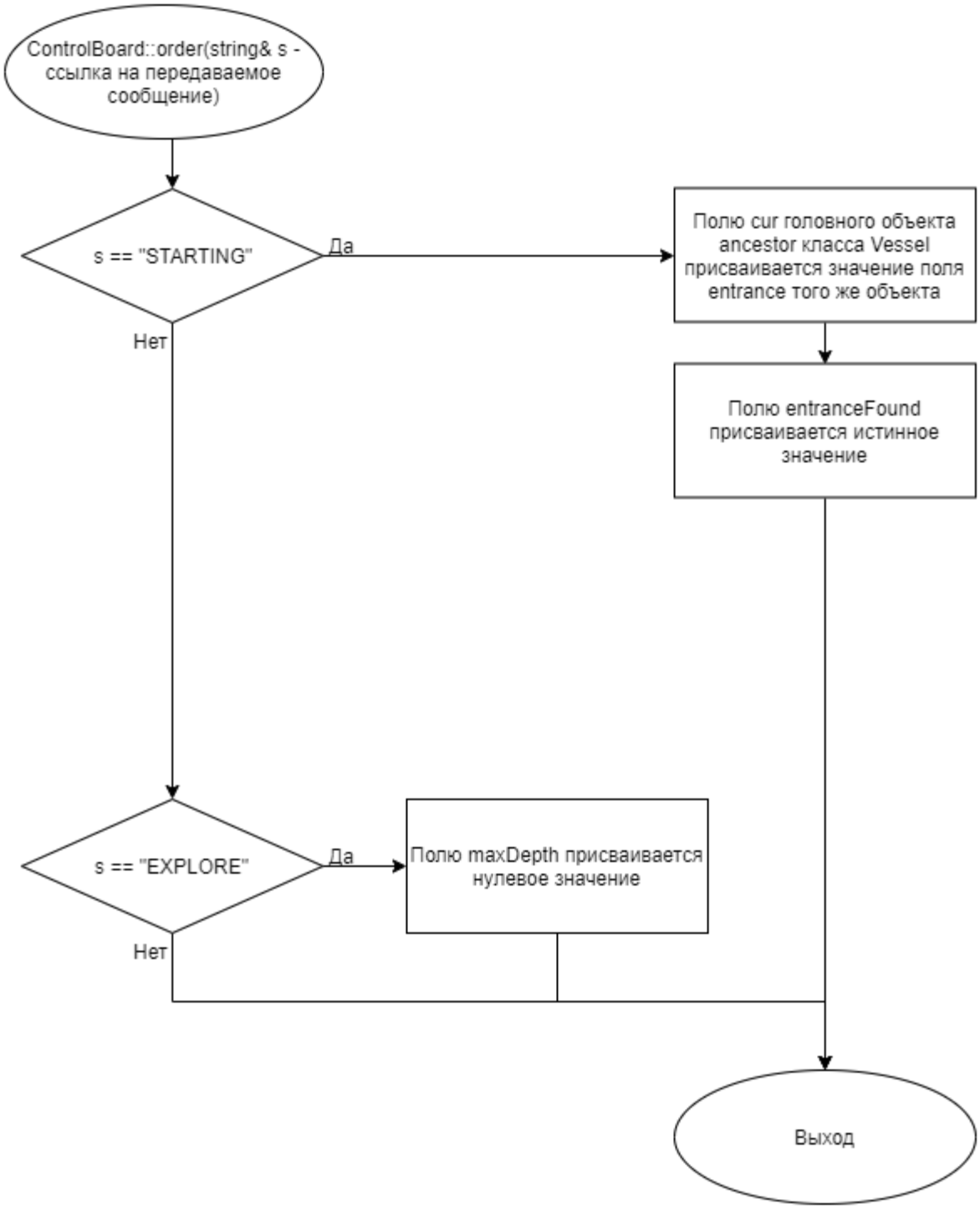

Рисунок 18 – Блок-схема алгоритма

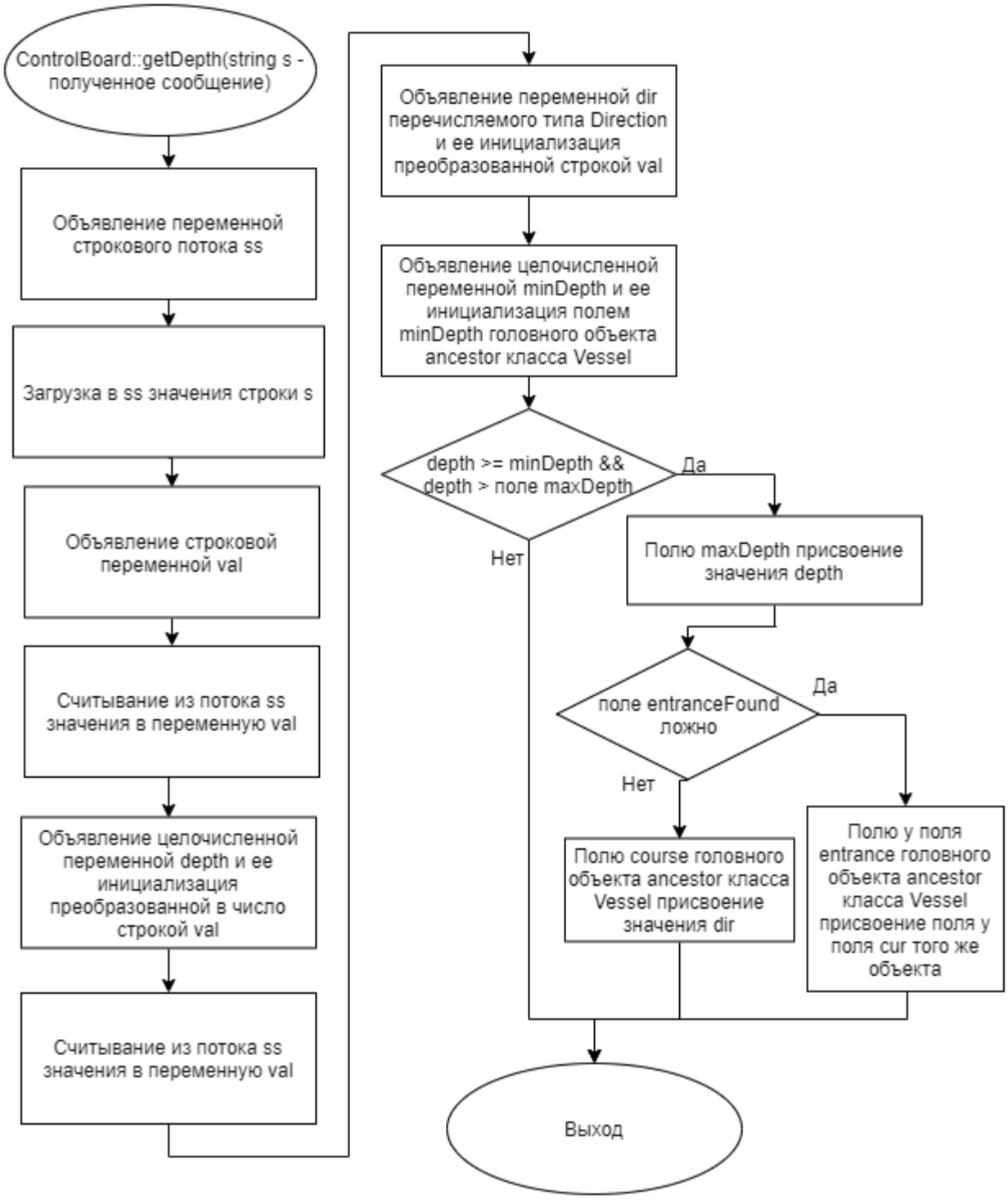

Рисунок 19 – Блок-схема алгоритма

Рисунок 20 – Блок-схема алгоритма

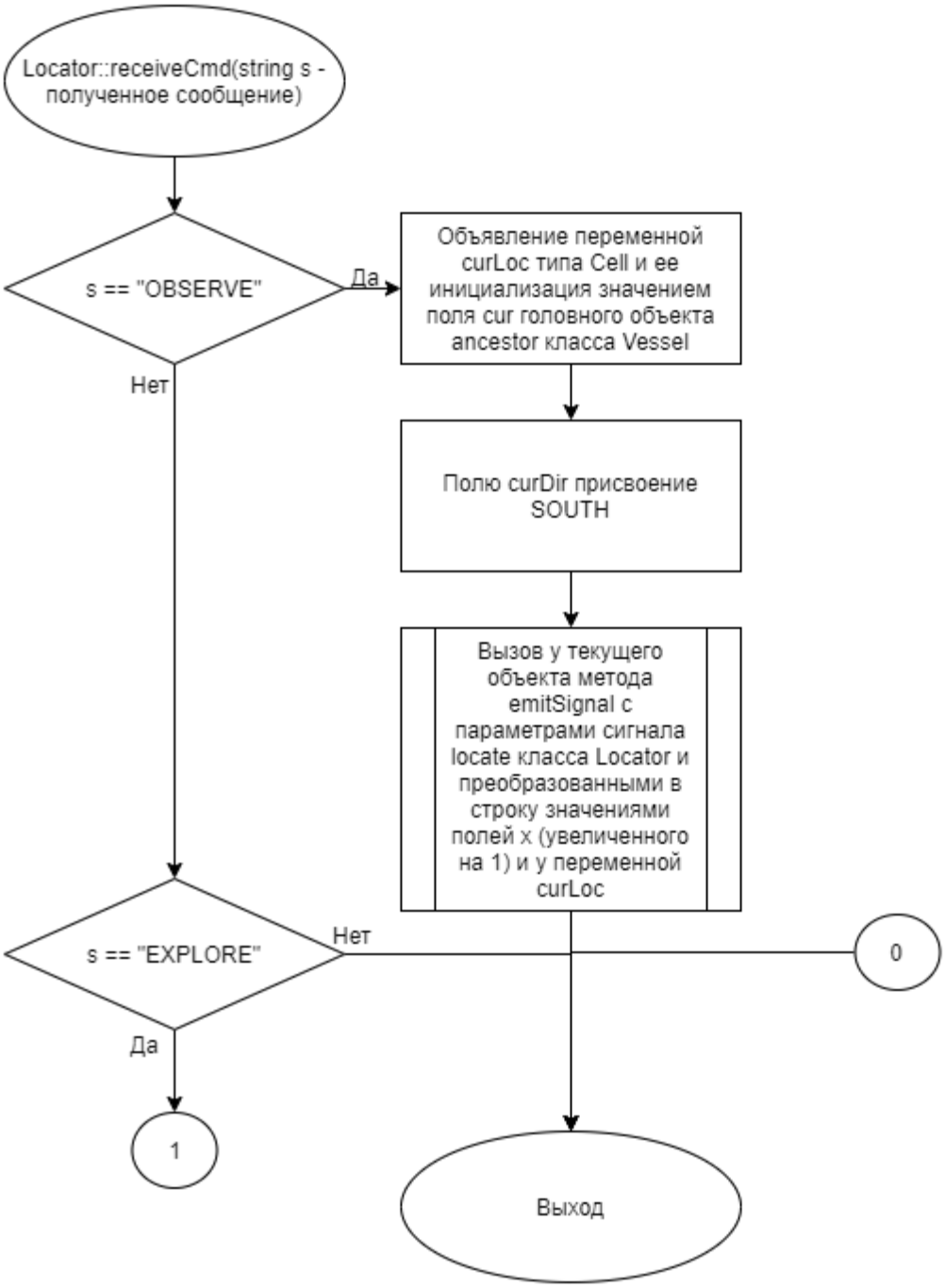

Рисунок 21 – Блок-схема алгоритма

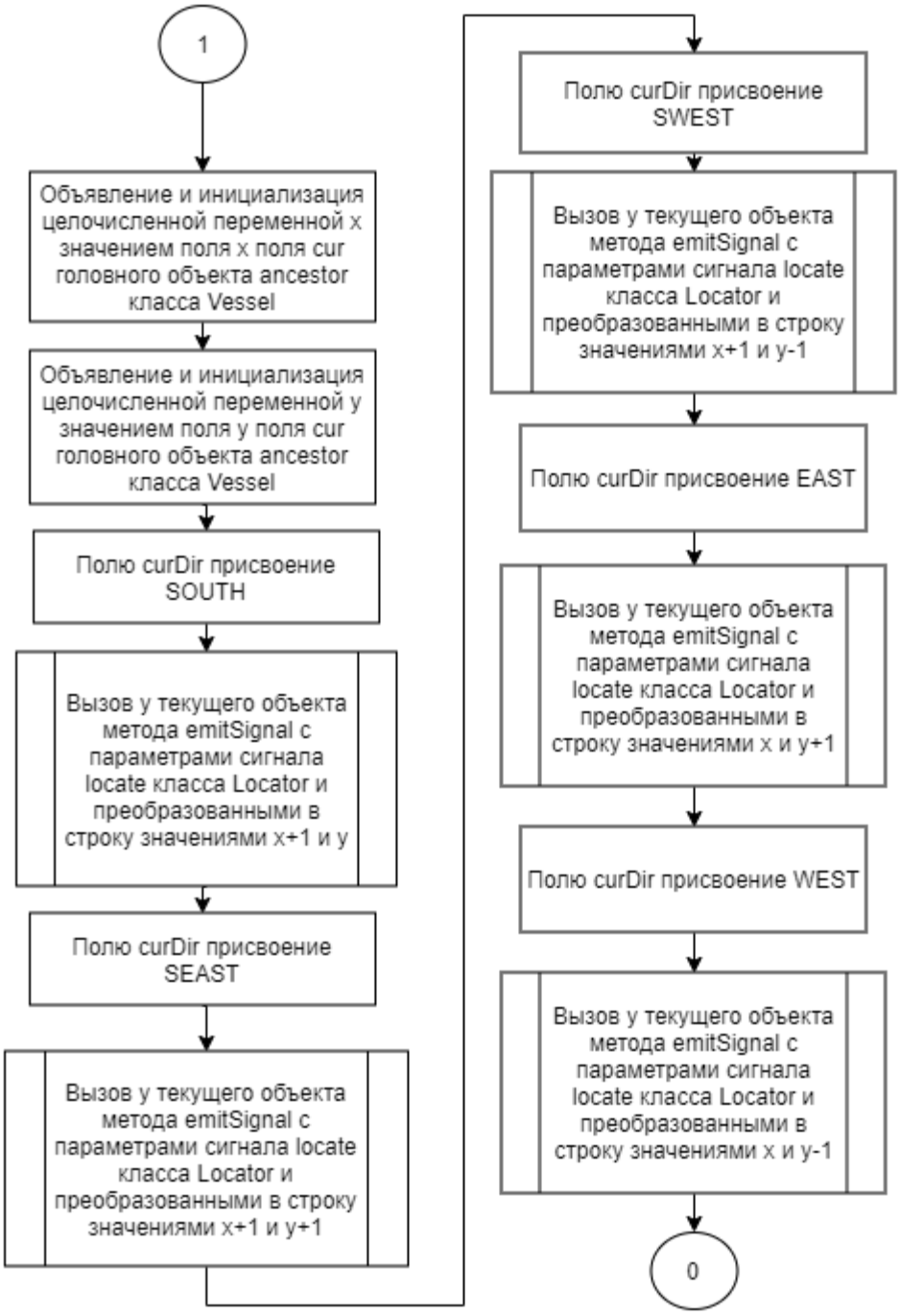

Рисунок 22 – Блок-схема алгоритма

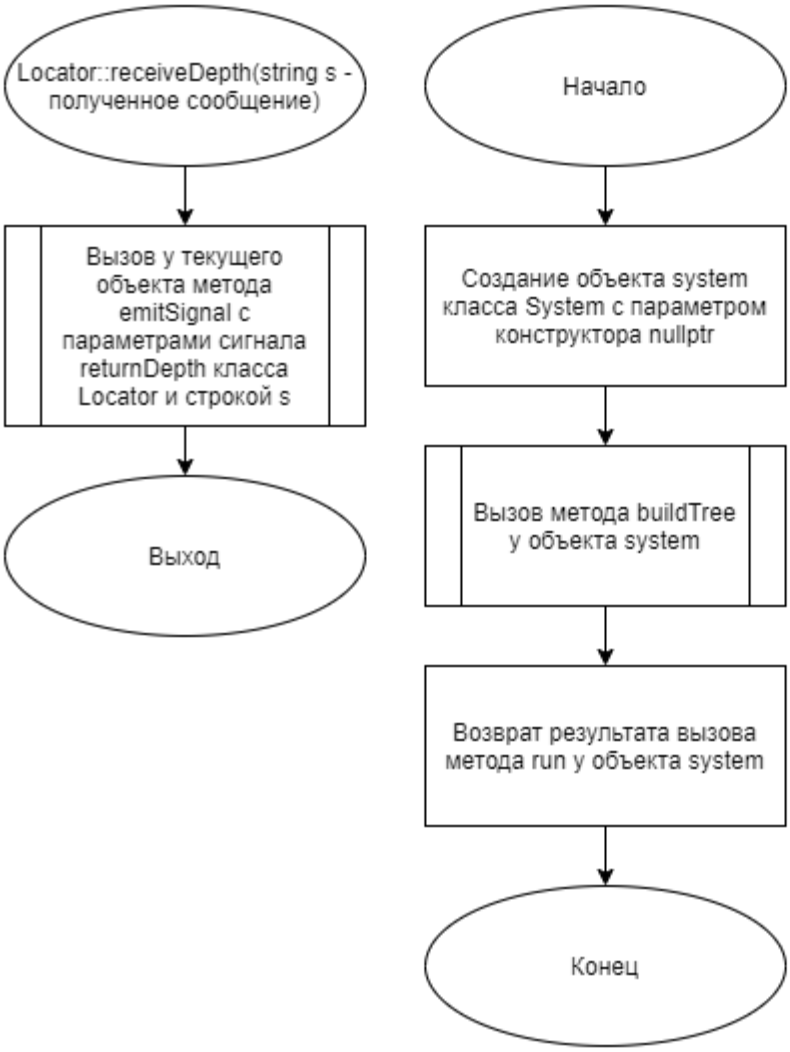

Рисунок 23 – Блок-схема алгоритма

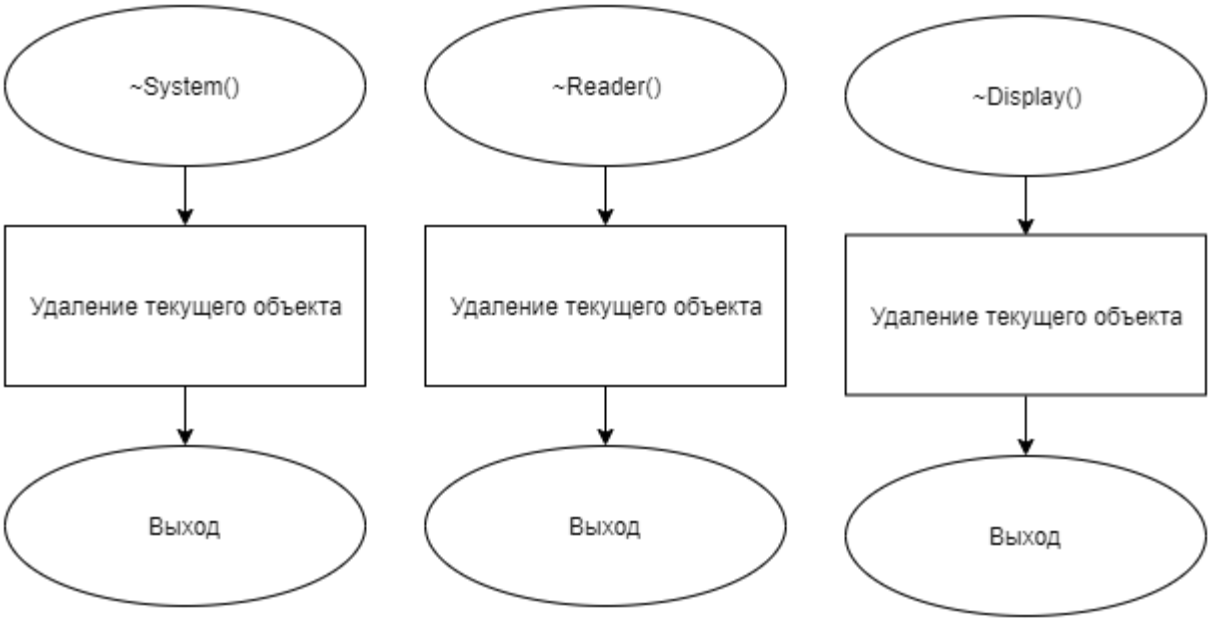

Рисунок 24 – Блок-схема алгоритма

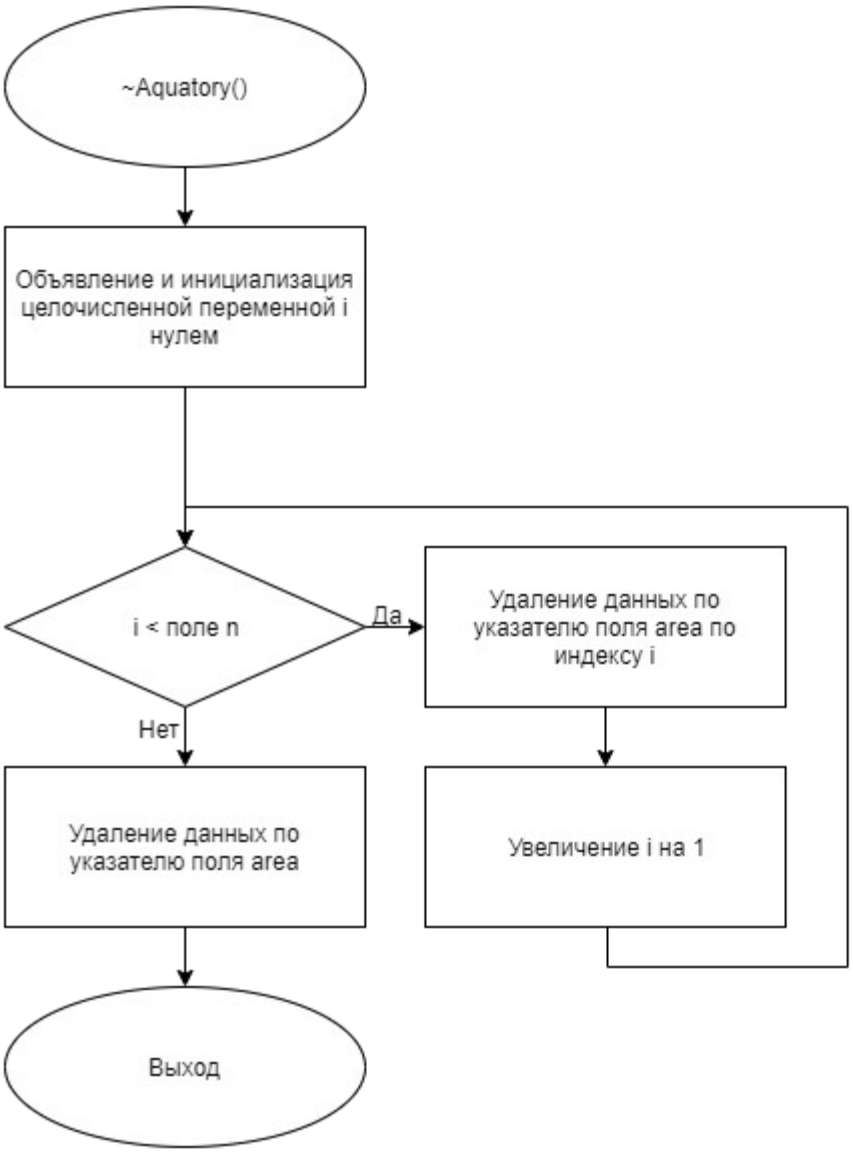

Рисунок 25 – Блок-схема алгоритма

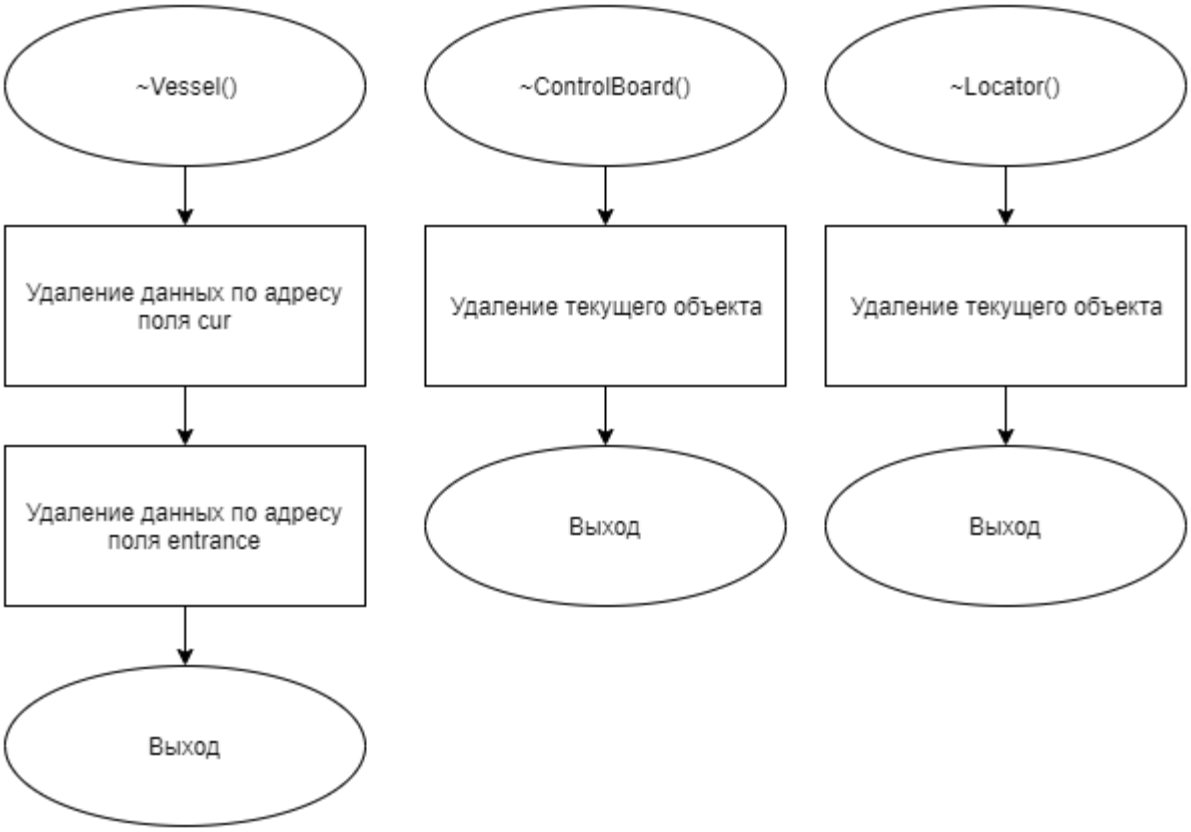

Рисунок 26 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.1 Файл Aquatory.cpp

Листинг 1 – Aquatory.cpp

```
#include "Aquatory.h"
#include "Vessel.h"

Aquatory::Aquatory(Base* ancestor, std::string name) : Base(ancestor, name)
{
   this->curLine = 0;
}

Aquatory::~Aquatory() { // Деструктор
   for(int i = 0; i < n; ++i) {
      delete this->area[i];
   }
   delete this->area;
}

void Aquatory::setProperties(std::string s) { // Обработчик для установки
параметров акватории
   std::stringstream ss;
   ss << s;
   std::string val;
   std::getline(ss, val, ' ');
   this->n = std::stoi(val);
   std::getline(ss, val, ' ');
   this->m = std::stoi(val);

   //if(n <= 10 || m <= 10) exit(-1);

   this->area = new int*[n];
   for(int i = 0; i < n; ++i) this->area[i] = new int[m]; // Создание
целочисленной матрицы
}

void Aquatory::setLine(std::string s) { // Обработчик для установки глубин
   std::stringstream ss;
   ss << s;
   std::string val;

   for(int j = 0; j < this->m; ++j) { // Для всех клеток данной линии
акватории
```

```
      std::getline(ss, val, ' ');
      this->area[curLine][j] = std::stoi(val); // Установка глубины клетки
   }
   this->curLine++; // Переход к следующей строке
}

void Aquatory::returnDepth(std::string& s) {} // Сигнал возврата глубины
клетки

void Aquatory::getSignal(std::string s) { // Обработчик сигнала от локатора
   std::stringstream ss;
   ss << s;
   std::string val;
   int xl, yl;
   std::getline(ss, val, ' ');
   xl = std::stoi(val); // Координаты при отсчете с 1
   std::getline(ss, val);
   yl = std::stoi(val);
   if(xl >= 1 && xl <= this->n && yl >= 1 && yl <= this->m) { // Если такие
координаты есть в акватории
      this->emitSignal(SIGNAL(Aquatory::returnDepth),    std::to_string(this-
>area[xl-1][yl-1])); // Исследование клетки на глубину
   }
}
```

5.2 Файл Aquatory.h

Листинг 2 – Aquatory.h

```
#ifndef __AQUATORY__H
#define __AQUATORY__H
#include "Base.h" // Включение описания класса Base

class Aquatory : public Base {
private:
   int curLine; // Вспомогательный индекс текущей строки
public:
   int n, m; // Параметры акватории
   int** area; // Матрица глубин

   Aquatory(Base*   ancestor,   std::string   name);   //   Параметризированный
конструктор
   ~Aquatory(); // Деструктор

   void setProperties(std::string s); // Обработчик-установщик параметров
   void setLine(std::string s); // Обработчик-установщик глубин

   void returnDepth(std::string& s); // Сигнал локатору с значением глубины
   void getSignal(std::string s); // Обработчик сигнала локатора
};
```

```
#endif
```

5.3 Файл Base.cpp

Листинг 3 – Base.cpp

```
#include <iostream> // Включение стандартной библиотеки для работы с потоками ввода/вывода
#include <queue>
#include "Base.h" // Включение описания класса Base

Cell::Cell(int x, int y) : x(x), y(y) {}

Bond::Bond(TYPE_SIGNAL signal, Base* target, TYPE_HANDLER handler) : // Конструктор структуры связи
     signal(signal), target(target), handler(handler) {}

Base::Base(Base* ancestor, std::string name) { // Реализация параметризированного конструктора
   this->name = name; // Инициализация скрытого поля имени
   this->ancestor = ancestor; // Инициализация скрытого поля-указателя на головной объект
   if(this->ancestor != nullptr) { // Условие, что головной объект существует
      this->ancestor->successors.push_back(this); // Добавление головному объекту текущего объекта в качестве одного из подчиненных
   }
}

Base::~Base() {
   for(auto suc : this->successors) delete suc;
   for(auto bond : this->bonds) delete bond;
}

int Base::getNumber() { return 0; } // Возврат номера текущего класса

bool Base::rename(std::string newName) { // Реализация метода переименования объекта
   if(this->ancestor != nullptr) { // Условие, что головной объект существует
      for(auto successor : this->ancestor->successors) { // Проход в цикле for-each по элементам коллекции подчиненных объектов головного объекта
         if(successor->getName() == newName) return false; // Возврат ложного значения в случае совпадения нового имени с именем одного из братьев
      }
   }
   this->name = newName; // Присвоение нового имени текущему объекту
   return true; // Возврат истинного значения
}

bool Base::setAncestor(Base* newAncestor) { // Реализация метода
```

```
переустановки головного объекта
   if(this->ancestor == nullptr) return false; // Если это корневой объект
   if(newAncestor->getSuccessor(this->name) != nullptr) return false; // Если
у нового головного уже есть подчиненный с таким именем

   Base* temp = newAncestor; // Обход предков нового головного
   while(temp->ancestor != nullptr) {
      if(temp == this) return false; // Если текущий объект - предок
головного, то менять нельзя
      temp = temp->ancestor;
   }

   this->ancestor->removeSuccessor(this->name); // Удалить текущий у прежнего
головного
   this->ancestor = newAncestor; // Установка нового головного
   newAncestor->successors.push_back(this); // Установка нового подчиненного
   return true;
}

std::string Base::getName() { // Реализация метода получения имени объекта
   return this->name; // Возврат значения имени объекта
}

std::string Base::getPath() { // Метод получения координаты объекта
   if(this->ancestor == nullptr) return "/";
   Base* temp = this;
   std::string path = ""; // Строковая переменная, восстанавливающая полную
координату удаленного объекта
   while(temp->ancestor != nullptr) { // Пока не упремся в корневой объект
      path = "/" + temp->name + path;
      temp = temp->ancestor; // Вверх по иерархии
   }
   return path;
}

Base* Base::getAncestor() { // Реализация метода получения указателя на
головной объект
   return this->ancestor; // Возврат указателя на головной объект
}

Base* Base::getSuccessor(std::string name) { // Реализация метода получения
указателя на подчиненный объект по имени
   for(auto successor : this->successors) { // Проход в цикле for-each по
элементам коллекции подчиненных объектов
      if(successor->getName() == name) return successor; // Возврат указтеля
на подчиненный объект в случае совпадения имен
   }
   return nullptr; // Возврат нуль-указателя
}

Base* Base::find(std::string name) { // Реализация метода поиска объекта по
имени
   Base* res = nullptr; // Вспомогательный указатель на объект
   std::queue<Base*> q; // Очередь для реализации обхода дерева
   q.push(this); // Начинаем с текущего
```

```
   while(!q.empty()) { // Пока очередь не закончилась
      Base* front = q.front(); // Рассматриваем следующий эл-т
      q.pop(); // Удаляем его из очереди

      if(front->getName() == name) { // Если его имя - искомое
         if(res == nullptr) res = front; // Если имя еще не встречалось
         else return nullptr; // Два объекта с одним именем
      }
      for(auto successor : front->successors) { // Проход по подчиненным
рассматриваемого
         q.push(successor); // Кладем их в очередь
      }
   }
   return res;
}

Base* Base::globalSearch(std::string name) { // Реализация метода поиска
объекта по имени во всем дереве иерархии
   Base* head = this; // Создание временного объекта для поиска корня дерева
   while(head->getAncestor() != nullptr) head = head->getAncestor(); // Поиск
корня дерева
   return head->find(name); // Возврат результата вызова метода find у
корневого элемента
}

Base* Base::getObject(std::string s_path) { // Реализация метода поиска
объекта по координате
   int size = s_path.size(); // Размер строки с координатой
   if(size == 0) return nullptr; // Если строка пуста

   Base* temp = this; // Вспомогательный объект
   std::string name; // Вспомогательная строка
   std::stringstream path; // Для удобства обработки координат используем
строковый поток
   path << s_path; // Записываем в поток исходную строку

   if(path.peek() == '/') { // Смотрим первый эл-т потока
      path.get(); // Пропускаем первый эл-т потока
      if(size == 1) { // Если строка из одного эл-та
         while(temp->ancestor != nullptr) { // Пока не упремся в корневой
элемент
            temp = temp->getAncestor(); // Переход к предыдущему в иерархии
         }
         return temp; // Возврат значения вспомогательного объекта
      } else if(path.peek() == '/') { // Смотрим первый эл-т потока
         path.get(); // Пропускаем первый эл-т потока
         std::getline(path, name); // Считывание имени следующего объекта в
координате
         return this->globalSearch(name); // Поиск по всему дереву
      } else {
         temp = this; /**/
         while(temp->ancestor != nullptr) {  // Пока не упремся в корневой
элемент
            temp = temp->getAncestor();  // Переход к предыдущему в иерархии
```

```
            }
            std::getline(path, name, '/'); // Считывание имени следующего
объекта в координате
            while(path && temp) { // Пока поток не пуст
               temp = temp->getSuccessor(name); // Переход к следующему в
иерархии
               std::getline(path, name, '/'); // Считывание имени следующего
объекта в координате
            }
            return temp; // Возврат значения вспомогательного объекта
         }
      } else if(path.peek() == '.') { // Смотрим первый эл-т потока
         path.get(); // Пропускаем первый эл-т потока
         if(size == 1) { // Если строка из одного эл-та
            return this; // Вернуть текущий объект
         } else {
            std::getline(path, name); // Считывание имени следующего объекта в
координате
            return this->find(name); // Поиск по текущей ветке
         }
      } else {
         std::getline(path, name, '/'); // Считывание имени следующего объекта в
координате
         while(path) { // Пока поток не пуст
            temp = temp->getSuccessor(name); // Переход к следующему в иерархии
            std::getline(path, name, '/'); // Считывание имени следующего
объекта в координате
         }
         return temp; // Возврат значения вспомогательного объекта
      }
   }

   void Base::removeSuccessor(std::string name) { // Реализация метода удаления
   подчиненного объекта
      for(std::vector<Base*>::iterator it = this->successors.begin(); it !=
   this->successors.end();) { // Цикл с итератором по элементам коллекции
         if((*it)->name == name) { // Если такой объект найден
            it = this->successors.erase(it); // Удаление подчиненного из
   коллекции
         } else ++it; // Инкремирование итератора
      }

   }

   void Base::printTree() { // Реализации метода вывода иерархии объектов
      std::cout << "\n"; // Вывод перевода строки
      Base* temp = this->getAncestor(); // Вспомогательный объект для
   определения уровня объекта
      while(temp != nullptr) { // Пока не упремся в корневой элемент
         std::cout << "    "; // Вывод отступов
         temp = temp->getAncestor(); // Переход к предыдущему в иерархии
      }
      std::cout << this->getName(); // Вывод имени текущего объекта

      for(auto successor : this->successors) { // Проход в цикле for-each по
```

```
элементам коллекции подчиненных объектов
      successor->printTree();  //  Рекурсивный  вызов  текущего  метода  у
подчиненного объекта с новым отступом
   }
}

void Base::printStates() { // Реализации метода вывода иерархии объектов с
метками состояния
   //std::cout << "\n"; // Вывод перевода строки
   Base*  temp  =  this->getAncestor();  //  Вспомогательный  объект  для
определения уровня объекта
   while(temp != nullptr) { // Пока не упремся в корневой элемент
      std::cout << "    "; // Вывод отступов
      temp = temp->getAncestor(); // Переход к предыдущему в иерархии
   }
   std::cout << this->getName() << " is"; // Вывод имени текущего объекта
   if(!this->state)  std::cout  <<  "  not";  //  Если  объект  отключен,  то
сигнализировать об этом
   std::cout << " ready\n";
   for(auto successor : this->successors) { // Проход в цикле for-each по
элементам коллекции подчиненных объектов
      successor->printStates();  //  Рекурсивный  вызов  текущего  метода  у
подчиненного объекта с новым отступом
   }
}

void Base::setState(int state) { // Реализация метода установки состояния
объекта
   if(state) { // Если состояние ненулевое
      bool fl = true; // Инициализация вспомогательной булевской переменной
истинным значением
      Base* ptr = this->getAncestor(); // Создание временного объекта для
прохода по всем предкам
      while(ptr != nullptr) { // Проход по всем предкам
         fl = fl && (bool)ptr->state; // Если у какого-то предка состояние
нулевое, то запомним это
         ptr = ptr->getAncestor();
      }
      if(fl) this->state = state; // Если наш флаг все еще истинен, то можем
присваивать ненулевое состояние
   } else { // Если же состояние нулевое
      this->state = 0; // Присвоим нулевое состояние
      for(auto successor : this->successors) { // Проход в цикле for-each по
элементам коллекции подчиненных объектов
         successor->setState(0); // Установка им нулевых состояний
      }
   }
}

void Base::connect(TYPE_SIGNAL signal, Base* target, TYPE_HANDLER handler) {
// Реализация метода связи объектов
   for(auto bond : this->bonds) { // Проверяем, нет ли уже такой связи
      if(bond->signal == signal && bond->target == target && bond->handler ==
handler) return;
   }
```

```
   Bond* newBond = new Bond(signal, target, handler);
   this->bonds.push_back(newBond);// Записываем информацию о новой связи
}

void Base::disconnect(TYPE_SIGNAL signal, Base* target, TYPE_HANDLER
handler) { // Реализация метода разрыва связи объектов
   for(std::vector<Bond*>::iterator it = this->bonds.begin(); it != this-
>bonds.end();) {
      if((*it)->signal == signal && (*it)->target == target && (*it)->handler
== handler) { // Если такая связь имеется
         it = this->bonds.erase(it); // Удаляем запись о ней
      } else ++it;
   }
}

void Base::emitSignal(TYPE_SIGNAL signal, std::string message) {    //
Реализация метода выдачи сигнала от объекта
   if(this->state == 0) return; // Если объект отключен
   (this->*signal)(message); // Вызов метода-сигнала с сообщением message
   Base* cur_target;
   TYPE_HANDLER cur_handler;
   for(auto bond : this->bonds) { // Проход по всем связям
      if(bond->signal == signal) { // Если связь имеет данный сигнал
         cur_target = bond->target;
         cur_handler = bond->handler;
         if(cur_target->state)(cur_target->*cur_handler)(message);  //  Вызов
метода-обработчика с посланием message
      }
   }

}
```

5.4 Файл Base.h

Листинг 4 – Base.h

```
#ifndef __BASE__H // Директива препроцессора для одноразового включения
содержимого данного .h-файла
#define __BASE__H

#include <string> // Подключение стандартной библиотеки для работы со
строками
#include <vector> // Подключение стандартной библиотеки для работы с
контейнером vector
#include <sstream>

#define SIGNAL(func_signal_name) (TYPE_SIGNAL)(&func_signal_name) // Макрос
для приведения типа сигнала
#define HANDLER(func_handler_name) (TYPE_HANDLER)(&func_handler_name) //
Макрос для приведения типа обработчика
```

```
class Base;

typedef void (Base::*TYPE_SIGNAL) (std::string& s); // Объявление типа
сигнала
typedef void (Base::*TYPE_HANDLER) (std::string s); // Объявление типа
обработчика

enum Direction { // Перечисляемый тип для реализации направления движения
   EAST,
   SEAST,
   SOUTH,
   SWEST,
   WEST
};

struct Cell { // Тип данных для хранения координаты клетки акватории
   int x;
   int y;
   Cell(int x, int y);
};

struct Bond { // Структура связи
   TYPE_SIGNAL signal; // Хранимый тип сигнала
   Base* target; // Адресат
   TYPE_HANDLER handler; // Хранимый тип обработчика
   Bond(TYPE_SIGNAL signal, Base* target, TYPE_HANDLER handler);
};

class Base { // Базовый класс
private:
   std::string name; // Персональное имя объекта
   Base* ancestor; // Указатель на головной элемент
   std::vector<Base*> successors; // Динамический массив указателей на
подчиненные объекты
   int state = 0; // Состояние готовности объекта
   std::vector<Bond*> bonds;
public:
   Base(Base* ancestor, std::string name = "Base"); // Параметризированный
конструктор
   ~Base();

   virtual int getNumber(); // Метод получения номера класса
   bool rename(std::string newName); // Метод переименования объекта
   bool setAncestor(Base* newAncestor); // Метод установки нового головного
объекта
   std::string getName(); // Метод получения имени объекта
   std::string getPath(); // Метод получения координаты объекта
   Base* getAncestor(); // Метод получения указателя на головной объект
   Base* getSuccessor(std::string name); // Метод получения указателя на
подчиненный объект по его имени
   Base* find(std::string name); // Метод поиска объекта в данной ветке по
его имени
   Base* globalSearch(std::string name); // Метод поиска объекта во всем
дереве иерархии по его имени
   Base* getObject(std::string path); // Метод поиска объекта по его
```

```
координате

   void removeSuccessor(std::string name); // Удаление подчиненного по имени
   void setState(int state); // Метод установки состояния готовности объекта
   void printTree(); // Метод вывода иерархии объектов
   void printStates(); // Метод вывода иерархии объектов с метками готовности
объектов

   void connect(TYPE_SIGNAL signal, Base* target, TYPE_HANDLER handler); //
Метод установки связи между объектами
   void   disconnect(TYPE_SIGNAL   signal,   Base*   target,   TYPE_HANDLER
handler); // Метод разрыва связи между объектами
   void emitSignal(TYPE_SIGNAL signal, std::string message); // Метод выдачи
сигнала от объекта
};


#endif
```

5.5 Файл ControlBoard.cpp

Листинг 5 – ControlBoard.cpp

```
#include "Vessel.h"
#include "ControlBoard.h"

ControlBoard::ControlBoard(Base*    ancestor,    std::string    name)    :
Base(ancestor, name) {
   this->maxDepth = 0;
   this->entranceFound = false; // Состояние пульта - поиск входа в фарватер
}

ControlBoard::~ControlBoard() {} // Деструктор

void ControlBoard::order(std::string& s) { // s = {DRIFT FORWARD OBSERVE
EXPLORE STARTING}
   if(s == "STARTING") {
      *((Vessel*)    this->getAncestor())->cur    =    *((Vessel*)    this-
>getAncestor())->entrance; // Перемещение корабля в начало фарватера
      this->entranceFound = true; // Состояние пульта - проход по фарватеру
   } else if(s == "EXPLORE") {
      this->maxDepth = 0;
   }
}

void ControlBoard::getDepth(std::string s) {
   //Дешифровка данных
   std::stringstream ss;
   ss << s;
   std::string val;
```

```
   std::getline(ss, val, ' ');
   int depth = std::stoi(val);
   std::getline(ss, val);
   Direction dir = (Direction)std::stoi(val);
   int minDepth = ((Vessel*) this->getAncestor())->minDepth; // Минимально
допустимая глубина

   if(depth >= minDepth && depth > this->maxDepth) { // Если данная глубина
удовлетворяет проходу
      this->maxDepth = depth;
      if(!this->entranceFound) {
         ((Vessel*) this->getAncestor())->entrance->y = ((Vessel*) this-
>getAncestor())->cur->y; // Установка координаты вхождения
      } else {
         ((Vessel*) this->getAncestor())->course = dir; // Установка штурвала
      }
   }
}
```

5.6 Файл ControlBoard.h

Листинг 6 – ControlBoard.h

```
#ifndef __CONTROLBOARD__H
#define __CONTROLBOARD__H
#include "Base.h" // Включение описания класса Base

class ControlBoard : public Base {
private:
   int maxDepth; // Значение максимальной глубины из все рассмотренных на
данном этапе
   bool entranceFound; // Состояние пульта
public:
   ControlBoard(Base* ancestor, std::string name); // Параметризированный
конструктор
   ~ControlBoard(); // Деструктор

   void order(std::string& s); // Сигнал-команда
   void getDepth(std::string s); // Обработчик локатора

};


#endif
```

5.7 Файл Display.cpp

Листинг 7 – Display.cpp

```
#include<iostream>
#include "Display.h"

Display::Display(Base* ancestor, std::string name) : Base(ancestor, name) {}

Display::~Display() {}// Деструктор

void Display::print(std::string s) {
   if(s == "READ") return; // Если это команда для считывателя, то игнорируем
   std::cout << s << "\n"; // Вывод переданного сообщения на экран
}
```

5.8 Файл Display.h

Листинг 8 – Display.h

```
#ifndef __DISPLAY__H
#define __DISPLAY__H
#include "Base.h" // Включение описания класса Base

class Display : public Base {
public:
   Display(Base*   ancestor,   std::string   name);   //   Параметризированный
конструктор
   ~Display(); // Деструктор

   void print(std::string s); // Обработчик сигналов для вывода
};


#endif
```

5.9 Файл Locator.cpp

Листинг 9 – Locator.cpp

```
#include "Vessel.h"
#include "Locator.h"
```

```
Locator::Locator(Base* ancestor, std::string name) : Base(ancestor, name) {
   this->curDir = EAST;
}

Locator::~Locator() {} // Деструктор

void Locator::locate(std::string& s) {} // Сигнал акватории

void Locator::returnDepth(std::string& s) { // Ответ пульту управления
   s += " " + std::to_string(this->curDir); // Добавление направления
рассматриваемой глубины
}

void Locator::receiveCmd(std::string s) { // Обработчик команд пульта
   if(s == "OBSERVE") { // Режим обозревания входов в фарватер
      Cell curLoc = * ((Vessel*) this->getAncestor())->cur;
      this->curDir = SOUTH;
      this->emitSignal(SIGNAL(Locator::locate), std::to_string(curLoc.x+1) +
" " + std::to_string(curLoc.y));
   } else if(s == "EXPLORE") { // Режим исследования глубин
      int x = ((Vessel*) this->getAncestor())->cur->x;
      int y = ((Vessel*) this->getAncestor())->cur->y;
      // Проверка всех доступных направлений движения
      this->curDir = SOUTH;
      this->emitSignal(SIGNAL(Locator::locate), std::to_string(x+1) + " " +
std::to_string(y));
      this->curDir = SEAST;
      this->emitSignal(SIGNAL(Locator::locate), std::to_string(x+1) + " " +
std::to_string(y+1));
      this->curDir = SWEST;
      this->emitSignal(SIGNAL(Locator::locate), std::to_string(x+1) + " " +
std::to_string(y-1));
      this->curDir = EAST;
      this->emitSignal(SIGNAL(Locator::locate), std::to_string(x) + " " +
std::to_string(y+1));
      this->curDir = WEST;
      this->emitSignal(SIGNAL(Locator::locate), std::to_string(x) + " " +
std::to_string(y-1));
   }

}

void Locator::receiveDepth(std::string s) { // Обработка ответа от акватории
   this->emitSignal(SIGNAL(Locator::returnDepth), s);
}
```

5.10 Файл Locator.h

Листинг 10 – Locator.h

```
#ifndef __LOCATOR__H
```

```
#define __LOCATOR__H
#include "Base.h" // Включение описания класса Base

class Locator : public Base {
private:
   Direction curDir; // Рассматриваемое направление определения глубины
public:
   Locator(Base*   ancestor,   std::string   name);   //   Параметризированный
конструктор
   ~Locator(); // Деструктор

   void locate(std::string& s); // Сигналы для акватории
   void returnDepth(std::string& s); // Сигнал для пульта
   void receiveCmd(std::string s); // Обработчик команды от пульта
   void receiveDepth(std::string s); // Обработчик ответа от акватории

};


#endif
```

5.11 Файл main.cpp

Листинг 11 – main.cpp

```
#include "System.h" // Включение описания класса System

int main() // Главная функция программы, исполняемая компилятором
{
   System system(nullptr); // Создание объекта класса System
   system.buildTree(); // Вызов метода построения дерева иерархии объектов
   return system.run(); // Возврат результата работы основной части программы
}
```

5.12 Файл Reader.cpp

Листинг 12 – Reader.cpp

```
#include<iostream>
#include "Reader.h"

Reader::Reader(Base* ancestor, std::string name) : Base(ancestor, name) {}

Reader::~Reader() {} // Деструктор
```

```
void Reader::readln(std::string s) { // Обработчик команд системы
   if(s != "READ") return; // Работает только если система дала команду
чтения
   std::string info;
   std::getline(std::cin, info, '\n'); // Считывает строку данных
   this->emitSignal(SIGNAL(Reader::read), info); // Выдает сигнал со
считанными данными
}

void Reader::read(std::string& s) {} // Сигнал выдачи данных
```

5.13 Файл Reader.h

Листинг 13 – Reader.h

```
#ifndef __READER__H
#define __READER__H

#include "Base.h"

class Reader : public Base {
public:
   Reader(Base* ancestor, std::string name);
   ~Reader(); // Деструктор

   void readln(std::string s); // Обработчик команды от системы
   void read(std::string& s); // Сигнал со считанной инф-ей
};

#endif
```

5.14 Файл System.cpp

Листинг 14 – System.cpp

```
#include "System.h"
#include "Reader.h"
#include "ControlBoard.h"
#include "Aquatory.h"
#include "Vessel.h"
#include "Locator.h"
#include "Display.h"

System::System(Base* head) : Base(head) {}
```

```
System::~System() {} // Деструктор

void System::buildTree() { // Построение дерева иерархии объектов
   // Создание всех объектов для функционирования системы, а также установка
их характеристик
   this->rename("System");
   Reader* reader = new Reader(this, "Reader");
   Display* display = new Display(this, "Display");
   Aquatory* area = new Aquatory(this, "Aquatory");
   Vessel* vessel = new Vessel(this, "Vessel");
   ControlBoard* control = new ControlBoard(vessel, "ControlBoard");
   Locator* locator = new Locator(vessel, "Locator");

   this->setState(1); // Установка готовности системы
   reader->setState(1); // Установка готовности считывателя
   area->setState(1); // Установка готовности акватории
   vessel->setState(1); // Установка готовности корабля
   display->setState(1); // Установка готовности отображающего элемента

   this->connect(SIGNAL(System::order), reader, HANDLER(Reader::readln)); //
Установка связей для организации считывания данных
   reader->connect(SIGNAL(Reader::read),                                area,
HANDLER(Aquatory::setProperties));
   reader->connect(SIGNAL(Reader::read),                               vessel,
HANDLER(Vessel::setProperties));

   this->emitSignal(SIGNAL(System::order),  "READ");  //  Считывание  хар-к
акватории и корабля

   reader->disconnect(SIGNAL(Reader::read),                             area,
HANDLER(Aquatory::setProperties)); // Разрыв больше не нужных связей
   reader->disconnect(SIGNAL(Reader::read),                            vessel,
HANDLER(Vessel::setProperties));
   reader->connect(SIGNAL(Reader::read),                                area,
HANDLER(Aquatory::setLine)); // Установка связей для загрузки данных

   for(int i = 0; i < area->n; ++i) { // Считывание глубин клеток акватории
      this->emitSignal(SIGNAL(System::order), "READ"); // Команда на чтение
строки
   }

   reader->disconnect(SIGNAL(Reader::read),                             area,
HANDLER(Aquatory::setLine));

   // Установка связей, необходимых для взаимодействия объектов друг с другом
   control->connect(SIGNAL(ControlBoard::order),                       locator,
HANDLER(Locator::receiveCmd));
   control->connect(SIGNAL(ControlBoard::order),                        vessel,
HANDLER(Vessel::move));
   locator->connect(SIGNAL(Locator::returnDepth),                      control,
HANDLER(ControlBoard::getDepth));
   locator->connect(SIGNAL(Locator::locate),                            area,
HANDLER(Aquatory::getSignal));
   area->connect(SIGNAL(Aquatory::returnDepth),                        locator,
```

```
HANDLER(Locator::receiveDepth));
   reader->connect(SIGNAL(Reader::read),                                    this,
HANDLER(System::cmdProcesser));
   this->connect(SIGNAL(System::order), display, HANDLER(Display::print));
   vessel->connect(SIGNAL(Vessel::printFairway),                        display,
HANDLER(Display::print));

   this->emitSignal(SIGNAL(System::order), "Ready to work"); // Сигнализация
о готовности системы

}

int System::run() { // Обеспечение отработки тактов
   (this->find("Locator"))->setState(1); // Установка готовности локатора
   Aquatory* area = (Aquatory*) this->find("Aquatory");
   Vessel* vessel = (Vessel*) this->find("Vessel");
   ControlBoard* control = (ControlBoard*) this->find("ControlBoard");
   control->setState(1); // Установка готовности пульта управления


   //Поиск входа в фарватер
   for(int i = 0; i < area->m; ++i) { // Каждая итерация - один такт
      control->emitSignal(SIGNAL(ControlBoard::order), "OBSERVE"); // Команда
исследования глубин
      control->emitSignal(SIGNAL(ControlBoard::order),  "DRIFT");  //  Команда
для движения корабля вдоль первой строки
   }

   control->emitSignal(SIGNAL(ControlBoard::order),  "STARTING");  //  Команда
для подготовки лоцмана

   //Прохождение акватории с поиском фарватера
   bool isReach = false; // Маячок об успешном прохождении
   while(!isReach) { // Каждая итерация - один такт
      control->emitSignal(SIGNAL(ControlBoard::order), "EXPLORE"); // Команда
исследования глубин
      control->emitSignal(SIGNAL(ControlBoard::order), "FORWARD"); // Команда
для движения корабля

      if(vessel->cur->x  ==  area->n)  isReach  =  true;  //  Корабль  достиг
последней строки
   }
   vessel->emitSignal(SIGNAL(Vessel::printFairway), "Fairway"); // Сигнал для
печати фарватера
   this->emitSignal(SIGNAL(System::order),  "READ");  //  Проверка  на  наличие
команды SHOWTREE от пользователя
   this->emitSignal(SIGNAL(System::order),   "Turn   off   the   ATM");   //
Сигнализация о завершении работы системы
   return 0;
}

void System::order(std::string& s) {} // Сигнал-команда

void System::cmdProcesser(std::string s) {
   if(s == "SHOWTREE") { // Обработка команды вывода дерева иерархии
```

```
            this->printStates();
        }
    }
```

5.15 Файл System.h

Листинг 15 – System.h

```
    #ifndef __SYSTEM__H
    #define __SYSTEM__H

    #include "Base.h"

    class System : public Base {
    public:
        System(Base* ancestor);
        ~System(); // Деструктор

        void buildTree(); // Метод построения дерева иерархии объектов
        int run(); // Метод запуска работы основной части программы

        void order(std::string& s); // Сигнал с командой
        void cmdProcesser(std::string s); // Обработчик команд пользователя
    };

    #endif
```

5.16 Файл Vessel.cpp

Листинг 16 – Vessel.cpp

```
    #include "Vessel.h"
    #include<iostream>

    Vessel::Vessel(Base* ancestor, std::string name) : Base(ancestor, name) {
        // Инициализация свойств корабля
        this->cur = new Cell(1, 1);
        this->course = EAST;
        this->entrance = new Cell(2, -1);
        this->fairway = "";
    }

    Vessel::~Vessel() { // Деструктор
        delete this->cur;
        delete this->entrance;
```

```
}

void Vessel::printFairway(std::string& s) { // Сигнал для печати координат
пути (фарватера)
   s += " (1, " + std::to_string(this->entrance->y) + ")" + " (2, " +
std::to_string(this->entrance->y) + ")" + this->fairway; // Добавление к
фарватеру записей о координатах входа
}

void Vessel::setProperties(std::string s) { // Обработчик установки
характеристик
   std::stringstream ss;
   ss << s;
   std::string val;
   std::getline(ss, val, ' '); // Игнорирование посторонней информации
   std::getline(ss, val, ' ');
   std::getline(ss, val);
   this->minDepth = std::stoi(val); // Установка минимально допустимой
глубины
}

void Vessel::move(std::string s) { // Обработчик движения корабля
   if(s == "DRIFT") {
      this->cur->y += 1; // Дрейф вдоль первой строки
   } else if(s == "FORWARD") {
      switch(this->course) { // Изменение координаты корабля в зависимости от
направления движения
      case EAST: {
         this->cur->y += 1;
         break;
      }
      case SEAST: {
         this->cur->y += 1;
         this->cur->x += 1;
         break;
      }
      case SOUTH: {
         this->cur->x += 1;
         break;
      }
      case SWEST: {
         this->cur->x += 1;
         this->cur->y -= 1;
         break;
      }
      case WEST: {
         this->cur->y -= 1;
         break;
      }
      }
      this->fairway += " (" + std::to_string(this->cur->x) + ", " +
std::to_string(this->cur->y) + ")"; // Запись новой координаты в фарватер
   }
}
```

5.17 Файл Vessel.h

Листинг 17 – Vessel.h

```
#ifndef __VESSEL__H
#define __VESSEL__H
#include "Base.h" // Включение описания класса Base

class Vessel : public Base {
private:
   std::string fairway; // Хранение координат пройденного фарватера
public:
   Cell* cur; // Текущие координаты расположения в акватории (отсчет с 1)
   Direction course; // Направление движения
   int minDepth; // Минимально допустимая глубина
   Cell* entrance; // Координаты вхождения в фарватер

   Vessel(Base*   ancestor,   std::string   name);   //   Параметризированный
конструктор
   ~Vessel(); // Деструктор

   void printFairway(std::string& s); // Сигнал для печати фарватера
   void setProperties(std::string s); // Обработчик-установщик значений
   void move(std::string s); // Обработчик сигнала от пульта
};


#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице 38.

Таблица 38 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
20 20 15 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 10 10 11 12 20 13 10 10 10 25 10 10 10 10 10 10 10 10 10 10 10 10 10 10 35 10 10 10 27 10 10 10 10 10 10 10 10 10 10 10 10 10 10 36 10 10 10 10 30 31 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 32 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 25 21 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 27 17 21 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 33 10 36 37 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 32 33 34 33 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 35 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 36 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 37 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 38 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 40 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10	Ready to work Fairway (1, 10) (2, 10) (3, 9) (4, 10) (5, 11) (6, 10) (7, 9) (8, 10) (9, 11) (9, 12) (10, 13) (11, 14) (12, 15) (13, 15) (14, 15) (15, 15) (16, 15) (17, 15) (18, 15) (19, 15) (20, 15) Turn off the ATM	Ready to work Fairway (1, 10) (2, 10) (3, 9) (4, 10) (5, 11) (6, 10) (7, 9) (8, 10) (9, 11) (9, 12) (10, 13) (11, 14) (12, 15) (13, 15) (14, 15) (15, 15) (16, 15) (17, 15) (18, 15) (19, 15) (20, 15) Turn off the ATM

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
41 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 43 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 44 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 45 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 46 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 47 10 10 10 10 10		
20 20 15 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 10 10 11 12 20 13 10 10 10 25 10 10 10 10 10 10 10 10 10 10 10 10 10 10 35 10 10 10 27 10 10 10 10 10 10 10 10 10 10 10 10 10 10 36 10 10 10 10 30 31 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 32 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 25 21 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 27 17 21 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 33 10 36 37 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 32 33 34 33 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 35 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 36 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10	Ready to work Fairway (1, 10) (2, 10) (3, 9) (4, 10) (5, 11) (6, 10) (7, 9) (8, 10) (9, 11) (9, 12) (10, 13) (11, 14) (12, 15) (13, 15) (14, 15) (15, 15) (16, 15) (17, 15) (18, 15) (19, 15) (20, 15) System is ready Reader is ready Display is ready Aquatory is ready Vessel is ready ControlBoard is ready Locator is ready Turn off the ATM	Ready to work Fairway (1, 10) (2, 10) (3, 9) (4, 10) (5, 11) (6, 10) (7, 9) (8, 10) (9, 11) (9, 12) (10, 13) (11, 14) (12, 15) (13, 15) (14, 15) (15, 15) (16, 15) (17, 15) (18, 15) (19, 15) (20, 15) System is ready Reader is ready Display is ready Aquatory is ready Vessel is ready ControlBoard is ready Locator is ready Turn off the ATM

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
37 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 38 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 40 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 41 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 43 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 44 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 45 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 46 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 47 10 10 10 10 10 SHOWTREE		
20 20 15 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 10 10 10 10 10 10 10 10 10 15 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10	Ready to work Fairway (1, 20) (2, 20) (3, 19) (4, 18) (5, 17) (6, 16) (7, 17) (8, 16) (9, 15) (10, 14) (11, 14) (12, 15) (13, 15) (14, 15) (15, 15) (16, 15) (17, 15) (18, 15) (19, 15) (20, 15) Turn off the ATM	Ready to work Fairway (1, 20) (2, 20) (3, 19) (4, 18) (5, 17) (6, 16) (7, 17) (8, 16) (9, 15) (10, 14) (11, 14) (12, 15) (13, 15) (14, 15) (15, 15) (16, 15) (17, 15) (18, 15) (19, 15) (20, 15) Turn off the ATM

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
10 10 10 20 20 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 30 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 20 10 10 10 10 10		

ЗАКЛЮЧЕНИЕ

В процессе выполнения работы было произведено моделирование системы, решающей задачу проведения корабля по фарватеру. Наряду с разработкой программного продукта велась сопроводительная документация. Достигнуты все поставленные задачи. А именно:

- Изучено объектно-ориентированное программирование на C++;
- На основе анализа требований к модели осуществлена программная реализация компонентов системы;
- Через комплекс сигналов и обработчиков налажено взаимодействие компонентов;
- Проведено тестирование программы. Результаты тестирования показали корректность работы программного продукта.

Таким образом, можем признать результаты проделанной работы удовлетворительными. Применение объектно-ориентированной парадигмы способствовало удобной и понятной разработке работоспособной системы.

В результате выполнения работы автор приобрел опыт поверсионной разработки информационных систем. Были получены навыки использования объектно-ориентированного подхода к программированию. Приобретенные за время выполнения курсовой работы знания и умения, безусловно, являются полезным подспорьем в профессиональной деятельности разработчика.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19 Единая система программной документации.

2. Методическое пособие студента для выполнения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratornyh_rabot_3.pdf (дата обращения 05.05.2021).

3. Приложение к методическому пособию студента по выполнению заданий в рамках курса «Объектно-ориентированное программирование» [Электронный ресурс]. URL: https://mirea.aco-avrora.ru/student/files/Prilozheniye_k_methodichke.pdf (дата обращения 05.05.2021).

4. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2019. — 624 с.

5. Видео лекции по курсу «Объектно-ориентированное программирование» [Электронный ресурс]. АСО «Аврора».

6. Антик М.И. Дискретная математика [Электронный ресурс]: Учебное пособие /Антик М.И., Казанцева Л.В. — М.: МИРЭА — Российский технологический университет, 2018 — 1 электрон. опт. диск (CD-ROM).