СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Знания объектно-ориентированного программирования и умение работать в этом стиле крайне востребованы для современного программиста. Концепция объектно-ориентированного программирования сейчас является самой распространенной. Данная концепция позволяет лучше структурировать код программы, отделяя общее от частного. Программа, использующая концепцию объектно-ориентированного программирования будет намного более понятной и удобочитаемой для автора и других разработчиков, в отличии от программы не использующей концепцию объектно-ориентированного программирования. Эта разница особенно заметна при написании объемных программ. Также подавляющее большинство компаний так или иначе требуют от своих разработчиков знания объектно-ориентированного программирования. Все вышеперечисленное явно дает понять, что для любого человека, который видит своей будущей профессией профессию программиста необходимо изучить концепцию объектно-ориентированного программирования и получить навыки работы с ней.

1 ПОСТАНОВКА ЗАДАЧИ

Выполнить моделирование работы системы следующей конструкции. Система содержит поле в виде квадратной матрицы клеток (позиций) 22 х 22. Каждая клетка поля либо заполнена, либо пустая. Координаты клетки меняются от 1 до 22 по горизонтали и по вертикали. Клетки в первой и последней строке, в первом и последнем столбце пустые (внешний контур). С помощью пустых клеток в квадрате с координатами от 2 и 21 по горизонтали и по вертикали заданы схемы лабиринтов. Имеется пульт управления и робот. Робот может:

Двигаться по сигналу (команде) от пульта вверх, вниз, влево и право на одну клетку, если она пустая;

Взаимодействовать с полем посредством сигнала и тот в ответ выдает сигнал с информацией о наличии пустых клеток относительно текущего положения робота;

Передать пульту управления сигнал необходимой согласно алгоритму информацией.

Перед запуском системы задается схема лабиринтов. Исходное положение робота координата (1, 2). Для поиска входа в лабиринт робот движется по часовой стрелке по пустым клеткам внешнего контура. Вход всегда изолирован, т.е. слева и справа, или сверху и снизу всегда присутствуют непустые клетки (стоят 1). Система ищет лабиринт и определяет, есть ли у лабиринта выход, отличный от входа или нет. Такой лабиринт может не существовать. На поле лабиринтов может быть несколько. Система завершает работу после обнаружения лабиринта с различными входом и выходом. Система функционирует по тактам, в рамках одного такта выполняется одно действие.

Построить систему, которая использует объекты:

1. Объект «система». Объект организует основной цикл тактов функционирования;

2. Объект для чтения данных. Считывает данные о схеме лабиринтов. После чтения очередной порции данных, объект выдает сигнал с текстом полученных данных. Все данные синтаксически корректны;
3. Объект пульта управления, для проведения анализа состояния системы и выдачи очередной команды посредством сигнала;
4. Объект, моделирующий поле размещения лабиринтов. Реагирует на получение сигнала с текущими координатами робота и сигналом возвращает информацию о соседних клетках. Пустой клетке соответствует значение 0, не пустой 1;
5. Объект, моделирующий устройство робота. Выполняет шаг по полю, принимает сигналы (команды) и делает запрос по сигналу объекту поле;
6. Объект для вывода результата отработки системы. Выводит информацию о найденных лабиринтах.

Все приведенные сигналы и соответствующие обработчики должны быть реализованы. 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

Первые 22 строки содержат схему расположения лабиринтов, которая задается посредством 0 и 1.

Пример ввода:

00000000000000000000

01011101111011111111110

01000111000000001111110

01011111111111111100000
01000111111101100001110
01111111111000011111110
01111111111111111111110
01111111111111111111110
01111111111111111111110
01110111111111111111110
01110111111111111110110
00000000001111111110110
01101111101111111110110
01101100000110000000000
01111111110000111110110
01111111111111110000110
01010110111111110111110
01010110111111110111110
00000000111111111111110
01111111111111111111110
01111111111111111111110
00000000000000000000000

1.2 Описание выходных данных

При обнаружении тупикового лабиринта выводиться координата входа по форме:

There is no way out of the maze («номер строки», «номер столбца»)

При обнаружении искомого лабиринта выводятся координаты входа и выхода по форме:

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

Пример вывода

There is no way out of the maze (1, 3)

There is no way out of the maze (1, 7)

There is no way out of the maze (1, 12)

There is no way out of the maze (4, 22)

Maze (14, 22) (12, 1)

2 МЕТОД РЕШЕНИЯ

Таблица 1 – Иерархия наследования классов

№	Имя Класса	Классы-наследники	Модификатор доступа при наследовании	Описание	Номер	Комментарий
1	Base			Базовый класс. Содержит основные поля и методы		
		Application	public		2	
		Reader	public		3	
		Brain	public		4	
		Field	public		5	
		Bot	public		6	
		Printer	public		7	
2	Application			Объект системы. Создает дерево связей и контролирует такты работы		

				системы.		
3	Reader			Объект чтения данных.		
4	Brain			Управляющий объект - пульт управления роботом.		
5	Field			Объект поля.		
6	Bot			Объект робота.		
7	Printer			Объект выдачи данных на печать.		

Изменения:

Application:

- изменения в buildTree
- изменения в startApp

Reader:

- метод insert функционал: ввод данных.

Brain

- Поля:
  - поле позиции входа в лабиринт по высоте:
    - Наименование - entryPosi
    - Тип - целочисленный

        - Модификатор доступа - закрытый
    - поле позиции входа в лабиринт по ширине:
        - Наименование - entryPosj
        - Тип - целочисленный
        - Модификатор доступа - закрытый
    - поле позиции входа в лабиринт по высоте для печати:
        - Наименование - entryPrintPosi
        - Тип - целочисленный
        - Модификатор доступа - закрытый
    - поле позиции входа в лабиринт по ширине для печати:
        - Наименование - entryPrintPosj
        - Тип - целочисленный
        - Модификатор доступа - закрытый
    - поле для проверки нахождния робота вне лабиринта:
        - Наименование - isOutside
        - Тип - логический
        - Модификатор доступа - закрытый
    - поле для проверки того, первый ли раз происходит вывод на печать:
        - Наименование - isFirstOutput
        - Тип - логический
        - Модификатор доступа - закрытый
- Методы:
    - findLabyrinthTact

      функционал: один такт работы программы по поиску лабиринта.

Field:

- Поля:
  - поле хранения поля:
    - Наименование - field
    - Тип - двумерный целочисленный массив
    - Модификатор доступа - закрытый
- Методы:
  - insertHandler

    функционал: обработчик сигнала ввода данных
  - getFieldStateHandler

    функционал: обработчик сигнала запроса получения данных о поле

Bot

- Поля:
  - поле позиции в лабиринте по высоте:
    - Наименование - posi
    - Тип - целочисленный
    - Модификатор доступа - закрытый
  - поле позиции в лабиринте по ширине:
    - Наименование - posj
    - Тип - целочисленный
    - Модификатор доступа - закрытый
  - поле направления взгляда робота:
    - Наименование - facing
    - Тип - символьный
    - Модификатор доступа - закрытый
- Методы:

- getFrontStateSignal

  функционал: сигнал для получения поля перед роботом
- checkForwardHandler

  функционал: обработчик сигнала запроса от пульта на получение данных о поле перед роботом
- moveForwardHandler

  функционал: обработчик команды движения вперед
- turnHandler

  функционал: обработчик команды поворота
- getCoordsHandler

  функционал: обработчик запроса от управления на получение положения
- absoluteMove

  функционал: движение на одну клетку в выбранном направлении

Printer

- printHandler

  функционал: вывод сообщения на печать

Base:

- emit_FSSignal

  функционал: испускание сигнала с возвратом данных

3 ОПИСАНИЕ АЛГОРИТМОВ

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

3.0 Алгоритм функции main

Функционал: представлена здесь лишь чтобы аврора создала обязательные блоки в блок-схеме.

Параметры: .

Возвращаемое значение: int.

Алгоритм функции представлен в таблице 2.

Таблица 2 – Алгоритм функции main

№	Предикат	Действия	№ перехода
1			∅

3.1 Алгоритм метода buildtree класса Application

Функционал: метод построения дерева.

Параметры: .

Возвращаемое значение: bool.

Алгоритм метода представлен в таблице 3.

Таблица 3 – Алгоритм метода buildtree класса Application

№	Предикат	Действия	№ перехода
1		создать объекты классов Reader, Brain, Field, Bot и Printer	2
2		установить все объекты системы в состояние готовности	3

№	Предикат	Действия	№ перехода
3		добавить нужные связи сигнал-обработчик, вызывая метод set_connection	4
4		вернуть true	∅

3.2 Алгоритм метода startApp класса Application

Функционал: метод работы системы.

Параметры: .

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 4.

Таблица 4 – Алгоритм метода startApp класса Application

№	Предикат	Действия	№ перехода
1		найти объекты reader, field и brain, вызывая метод findByPath	2
2		вызвать сигнал для ввода данных	3
3	был введен SHOWTREE		∅
			4
4		вызвать сигнал одного такта работы программы	5
5	по завершении такта было возвращено значения окончания работы программы		∅
			4

3.3 Алгоритм метода insert класса Reader

Функционал: сигнал ввода данных (ввод данных).

Параметры: string& field.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 5.

Таблица 5 – Алгоритм метода insert класса Reader

№	Предикат	Действия	№ перехода
1		field = пустая строка	2
2		ввод данных	3
3	введено SHOWTREE	field = SHOWTREE	∅
			4
4		записать введенные данные в field	∅

3.4 Алгоритм метода insertHandler класса Field

Функционал: обработка ввода.

Параметры: string fieldStr.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 6.

Таблица 6 – Алгоритм метода insertHandler класса Field

№	Предикат	Действия	№ перехода
1	fieldStr = SHOWTREE	вернуть "0"	∅
			2
2		заполнить двумерный массив field нулями и единицами из строки fieldStr	3
3		вернуть "1"	∅

3.5 Алгоритм метода getFieldStateHandler класса Field

Функционал: выдача информации о поле по переданным координатам.

Параметры: string posI100J.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 7.

Таблица 7 – Алгоритм метода getFieldStateHandler класса Field

№	Предикат	Действия	№ перехода
1		расшифровать координаты (переданное значение делим на сто и берем остаток от деления на сто)	2
2	координаты находятся за пределами поля	вернуть "-1"	∅
			3
3		вернуть строковую вариант содержимого массива field по индексам равным координатам	∅

3.6 Алгоритм метода printHandler класса Printer

Функционал: вывод на экран.

Параметры: string mes.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 8.

Таблица 8 – Алгоритм метода printHandler класса Printer

№	Предикат	Действия	№ перехода
1		вывод mes	2
2		вернуть пустую строку	∅

3.7 Алгоритм метода emit_FSSignal класса Base

Функционал: испускание сигнала.

Параметры: TYPE_SIGNAL signal, string posI100J.

Возвращаемое значение: int.

Алгоритм метода представлен в таблице 9.

Таблица 9 – Алгоритм метода emit_FSSignal класса Base

№	Предикат	Действия	№ перехода
1	объекта не существует или он не активен	вернуть -2	∅
		вызвать переданный сигнал с аргументом posI100J	2
2		вызват обработчик переданного сигнала в соответствии со связью	3
3		вернуть приведенное к целочисленному типу возвращенное вызванным обработчиком значение	∅

3.8 Алгоритм метода absolutemove класса Bot

Функционал: движение в выбранном направлении.

Параметры: char direction.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 10.

Таблица 20 – Алгоритм метода absolutemove класса Bot

№	Предикат	Действия	№ перехода
1		в зависимости от переданного direction (u, r, d или l) выбрать новую позицию на которую мы хотим перейти	2
2		вызвать сигнал проверки поля по координатам новой позиции	3
3	вызванный сигнал вернул 0	присвоить полям координат робота новые координаты (новой позиции)	∅
			∅

3.9 Алгоритм метода getFrontStateSignal класса Bot

Функционал: подготовка позиции для сигнала получения значения поля по координатам перед роботом.

Параметры: string& posI100J.

Возвращаемое значение: void.

Алгоритм метода представлен в таблице 11.

Таблица 31 – Алгоритм метода getFrontStateSignal класса Bot

№	Предикат	Действия	№ перехода
1		в зависимости от направления взгляда робота (facing: N, E, S или W) выбрть координаты клетки перед роботом	2
2		присвоить переменной posI100J зашифрованное значение координат (значение по высоте * 100 + значение по ширине)	∅

3.10 Алгоритм метода moveForwardHandler класса Bot

Функционал: обработка сигнала перемещения вперед.

Параметры: string placeholder.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 12.

Таблица 42 – Алгоритм метода moveForwardHandler класса Bot

№	Предикат	Действия	№ перехода
1		в зависимости от facing (N, E, S или W) выбрать новую позицию на которую мы хотим перейти	2
2		вызвать сигнал получения значения поля по координатам, передав новые координаты в зашифрованном виде	3
3	сигнал вернул 0	присвоить полям координат объекта новые	4

№	Предикат	Действия	№ перехода
		значения (новые координаты)	
			4
4		вернуть пустую строку	∅

3.11 Алгоритм метода checkForwardHandler класса Bot

Функционал: обработка запроса от управления на получение данных о поле перед роботом.

Параметры: string placeholder.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 13.

Таблица 53 – Алгоритм метода checkForwardHandler класса Bot

№	Предикат	Действия	№ перехода
1		вызвать emit_FSSignal передав сигнал getFrontStateSignal	2
2		вернуть то, что вернул emit_FSSignal	∅

3.12 Алгоритм метода turnHandler класса Bot

Функционал: обработчик команды поворота.

Параметры: string direction.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 14.

Таблица 64 – Алгоритм метода turnHandler класса Bot

№	Предикат	Действия	№ перехода
1	direction = "R"	присвоить facing новое значение (повернуть по часовой стрелки на 90 градусов)	2

№	Предикат	Действия	№ перехода
			2
2	direction = "L"	присвоить facing новое значение (повернуть против часовой стрелки на 90 градусов)	3
			3
3		вернуть пустую строку	∅

3.13 Алгоритм метода getCoordsHandler класса Bot

Функционал: обработчик запроса на получение координат робота.

Параметры: string placeholder.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 15.

Таблица 75 – Алгоритм метода getCoordsHandler класса Bot

№	Предикат	Действия	№ перехода
1		вернуть координаты объекта в зашифрованном формате	∅

3.14 Алгоритм метода findLabyrinthTact класса Brain

Функционал: выполнение одного такта поиска лабиринта.

Параметры: string placeholder.

Возвращаемое значение: string.

Алгоритм метода представлен в таблице 16.

Таблица 86 – Алгоритм метода findLabyrinthTact класса Brain

№	Предикат	Действия	№ перехода
1	робот находится на углу внешней дороги	отправить сигнал поворота направа и движения вперед	12

№	Предикат	Действия	№ перехода
			2
2		повернуть направо	3
3	впереди стена	повернуть налево	3
			4
4	робот поворачивал налево	идти вперед	6
		запомнить координаты, как координаты клетки перед входом	5
5		идти вперед	6
6	робот внутри	получить координаты робота	7
			12
7	робот находится на краю лабиринта		8
			19
8	позиция робота совпадает с координатами запомненного входа	вывод сообщения о том, что из лабиринта нет выхода	9
		идти вперед	10
9		идти вперед, повернуть направо, идти вперед	19
10		вывод сообщения о нахождении лабиринта	11
11		вернуть "0"	∅
12	робот поворачивал налево		19
		isOutside = false	13
13		получить координаты робота	14
14		запомнить координаты робота, как координаты входа	15
1	перед роботом стена	вывод сообщение о том, что из лабиринта не	16

№	Предикат	Действия	№ перехода
5		выхода	
			19
16		повернуть направо, повернуть направо, идти вперед	17
17		isOutside = true	18
18		повернуть направо, идти вперед	19
19	координаты робота равны (0, 0)	вернуть "0"	∅
		вернуть "1"	∅

4 БЛОК-СХЕМЫ АЛГОРИТМОВ

Представим описание алгоритмов в графическом виде на рисунках 1-14.

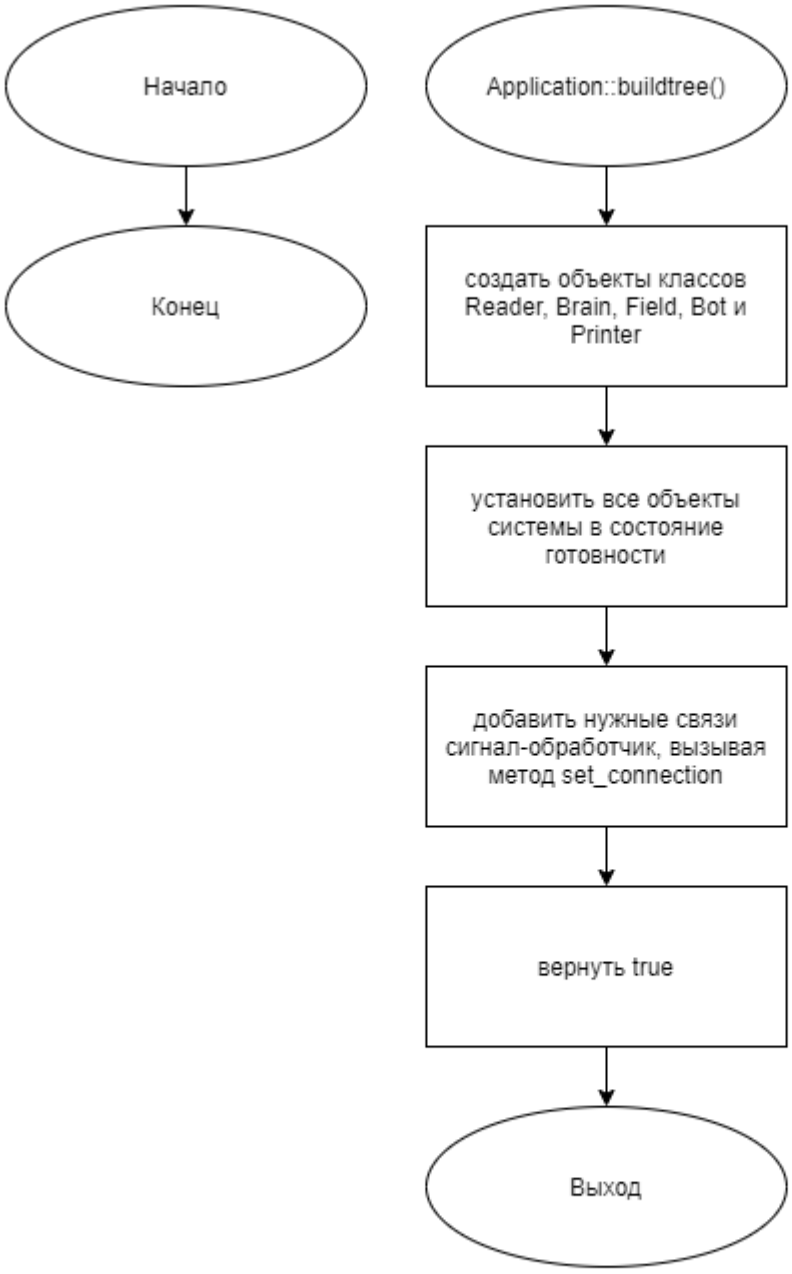

Рисунок 1 – Блок-схема алгоритма

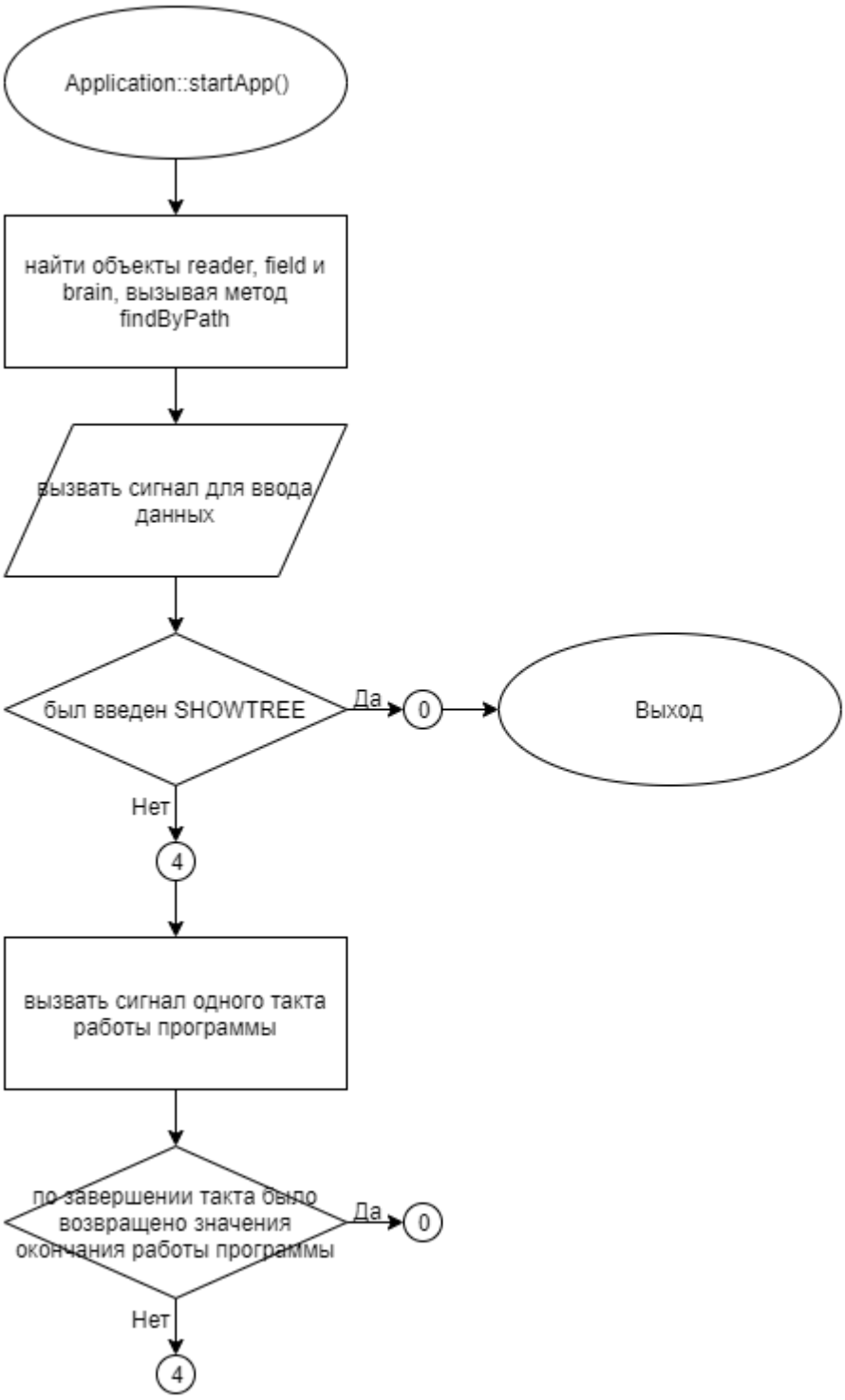

Рисунок 2 – Блок-схема алгоритма

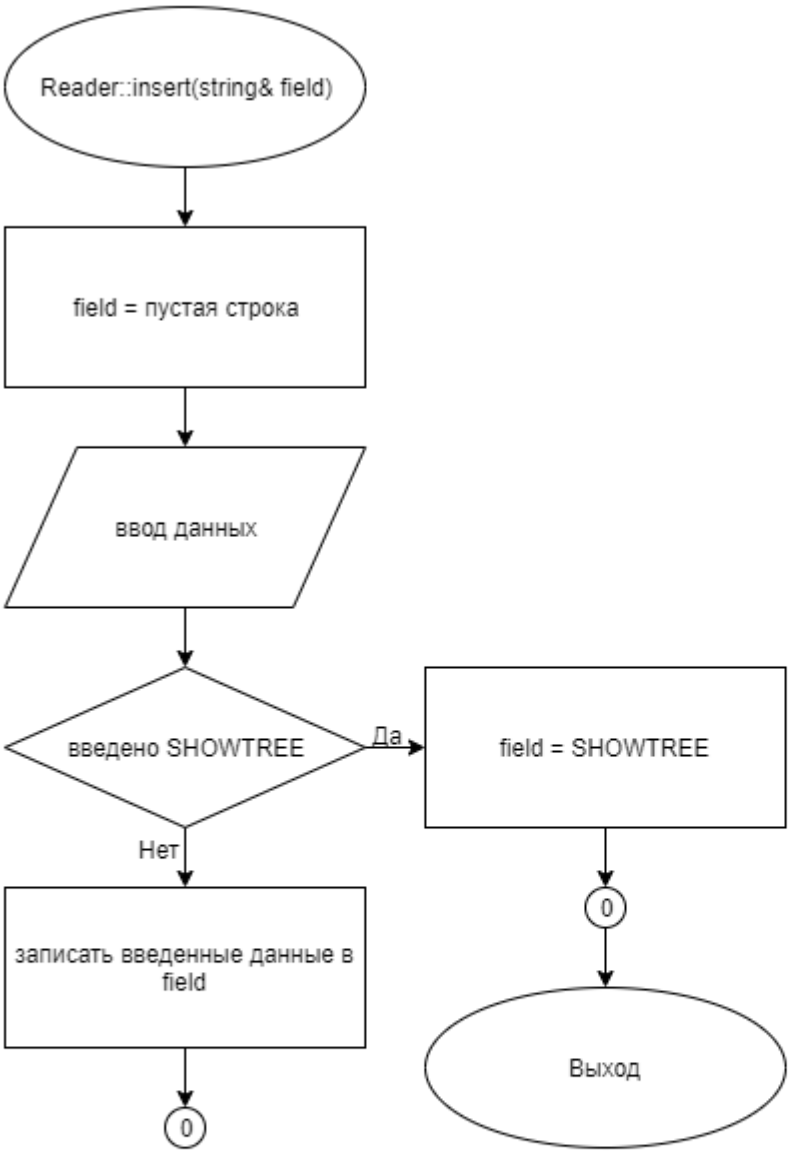

Рисунок 3 – Блок-схема алгоритма

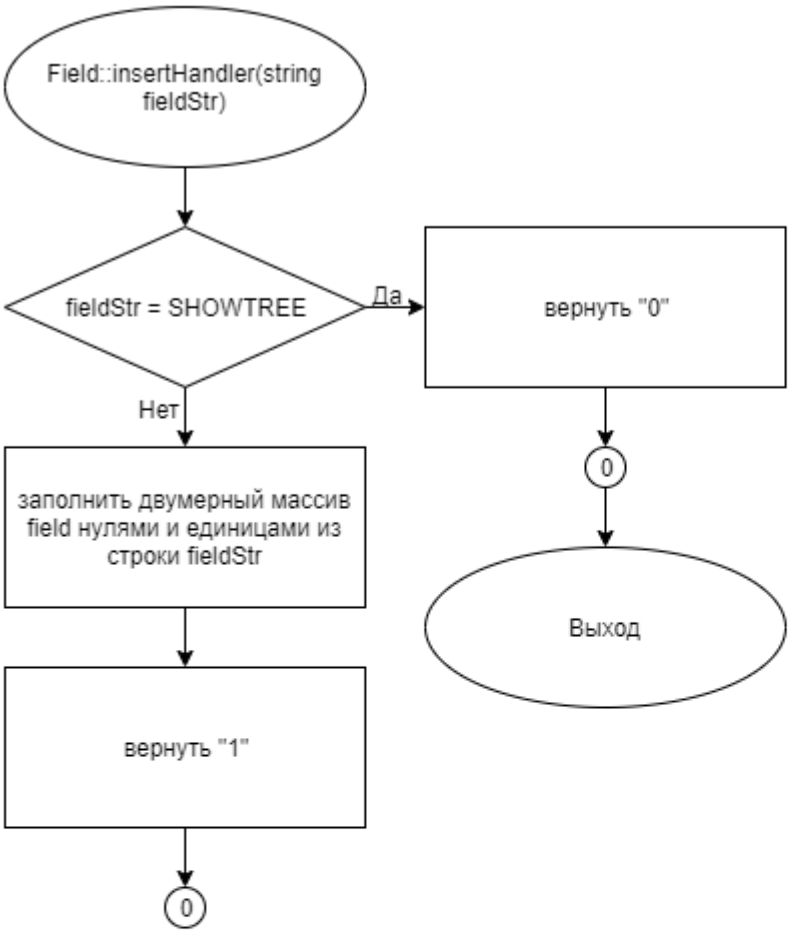

Рисунок 4 – Блок-схема алгоритма

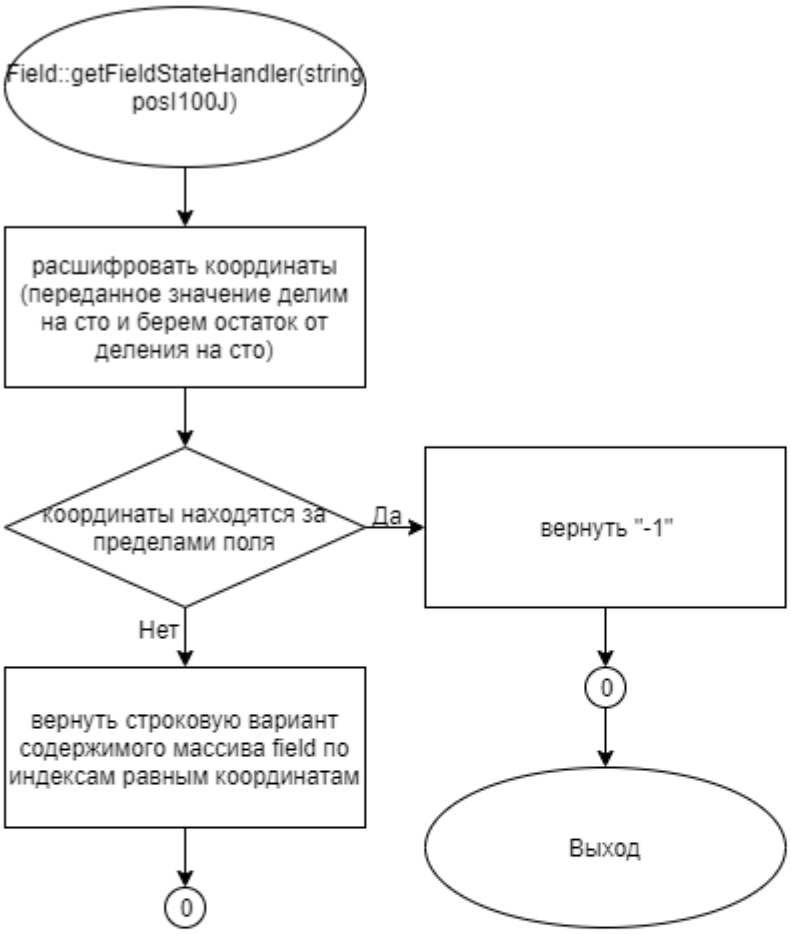

Рисунок 5 – Блок-схема алгоритма

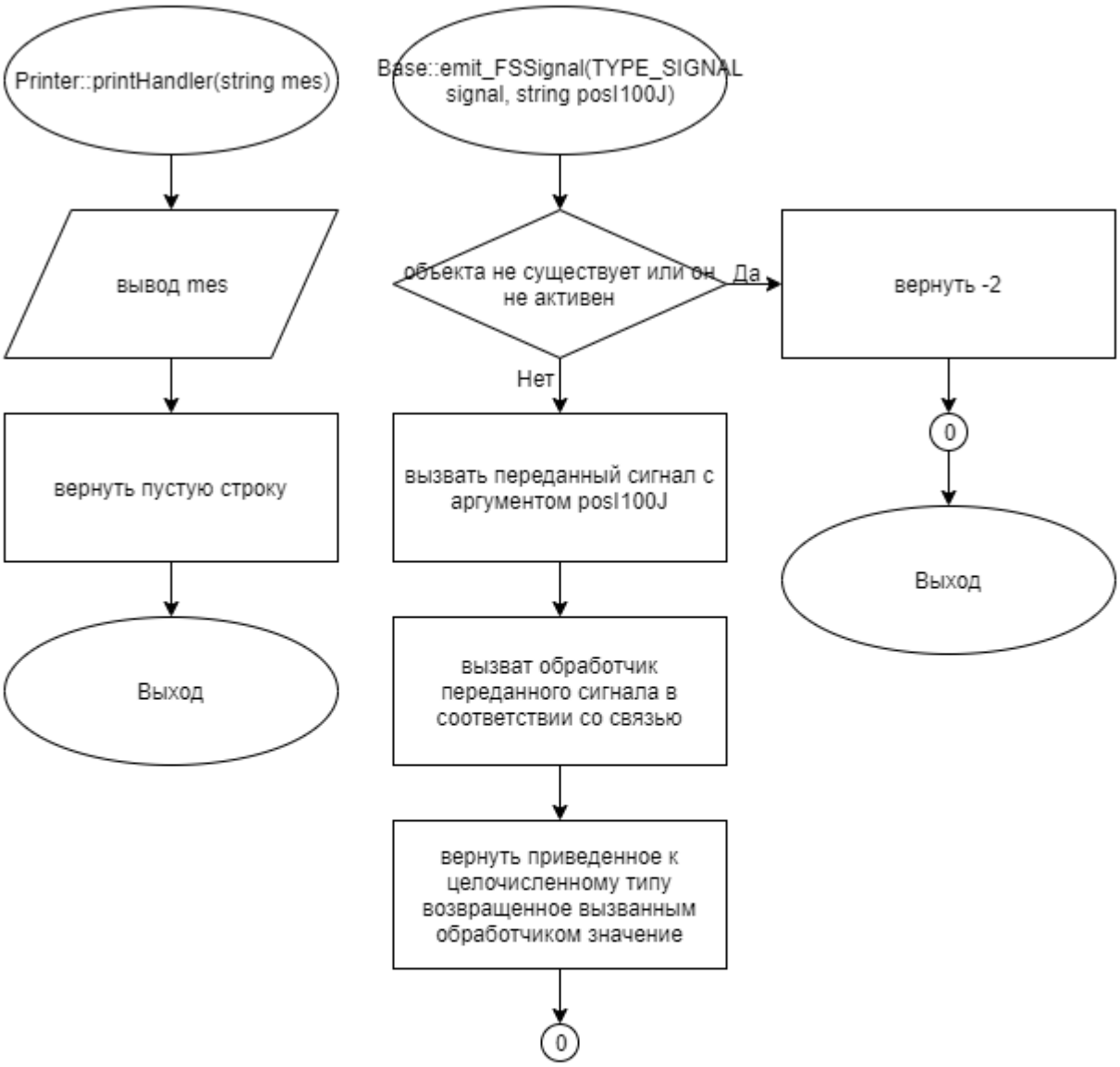

Рисунок 6 – Блок-схема алгоритма

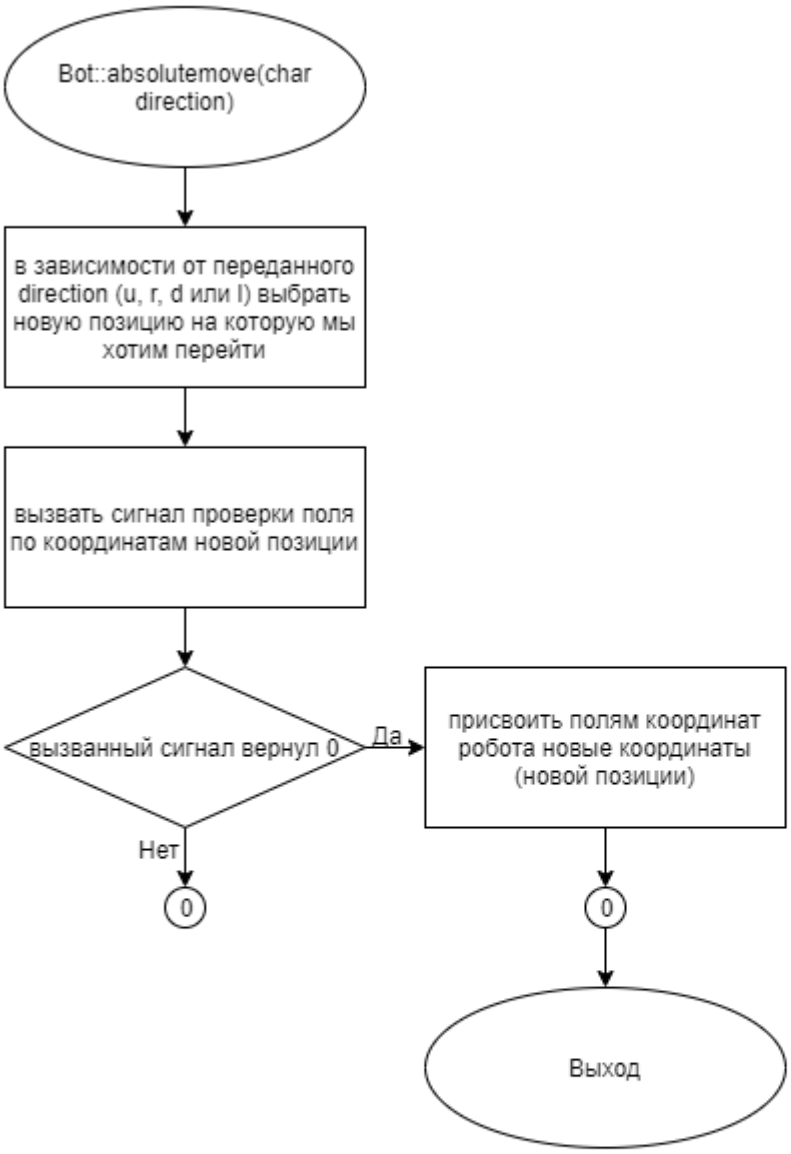

Рисунок 7 – Блок-схема алгоритма

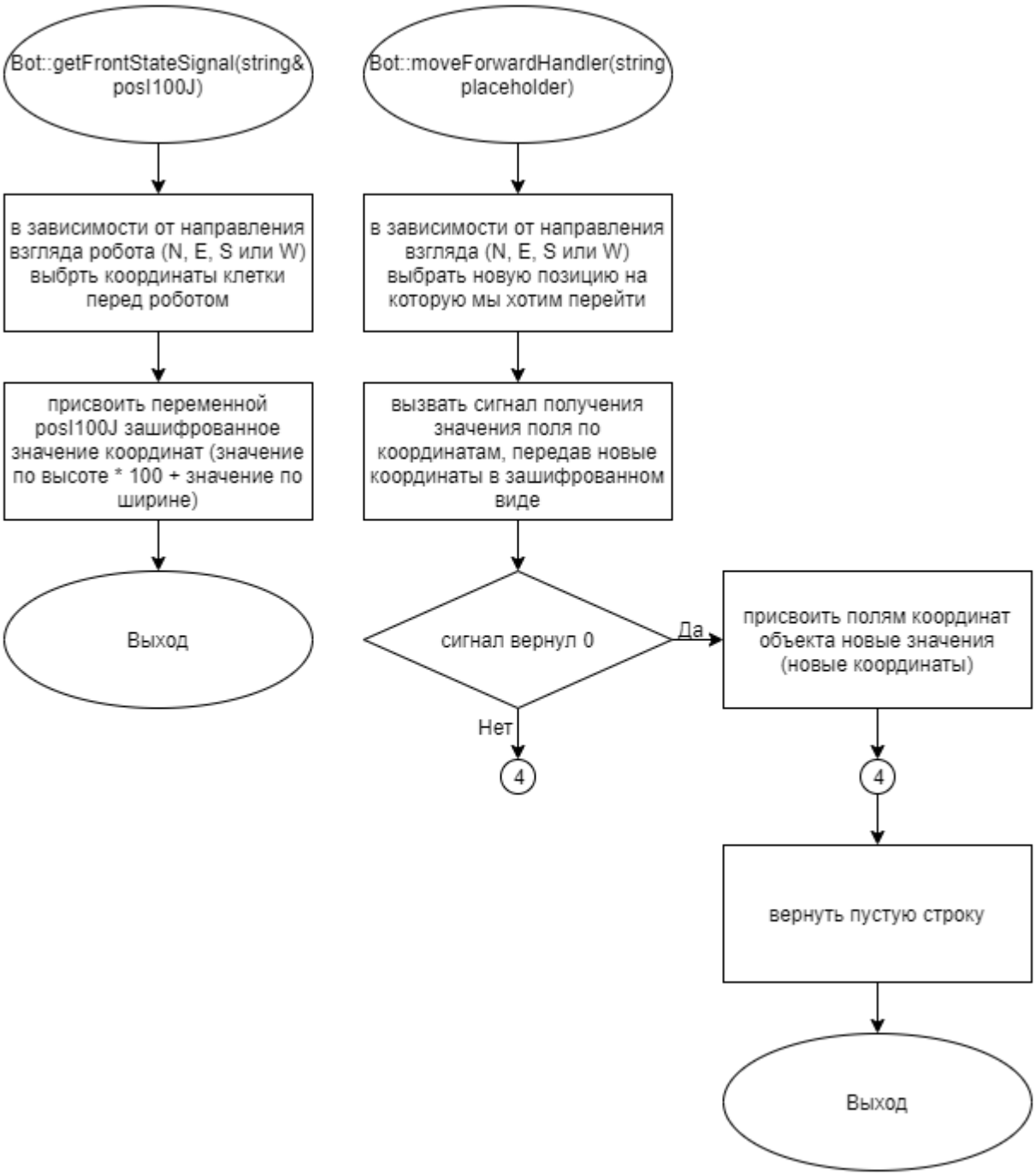

Рисунок 8 – Блок-схема алгоритма

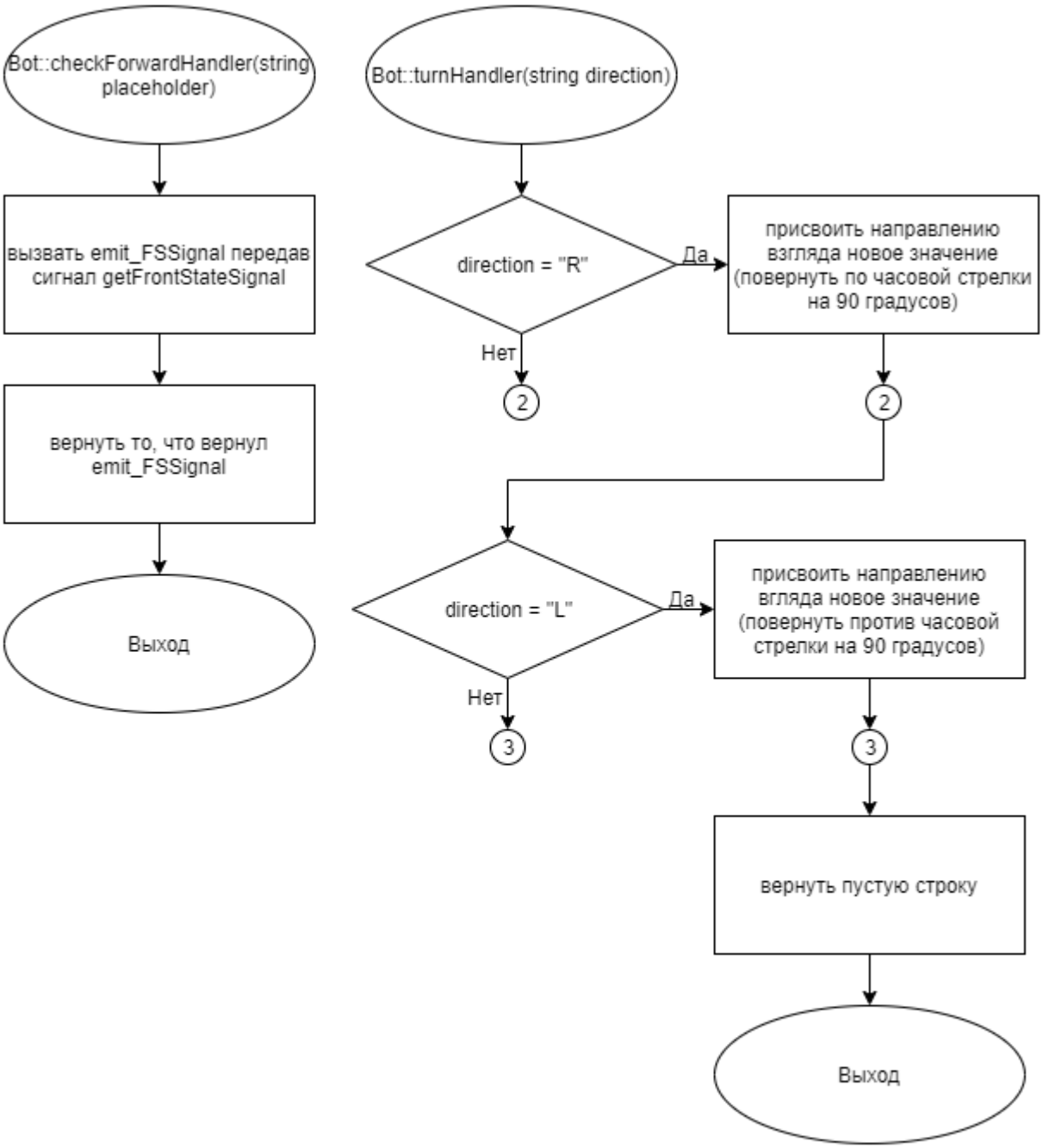

Рисунок 9 – Блок-схема алгоритма

Рисунок 10 – Блок-схема алгоритма

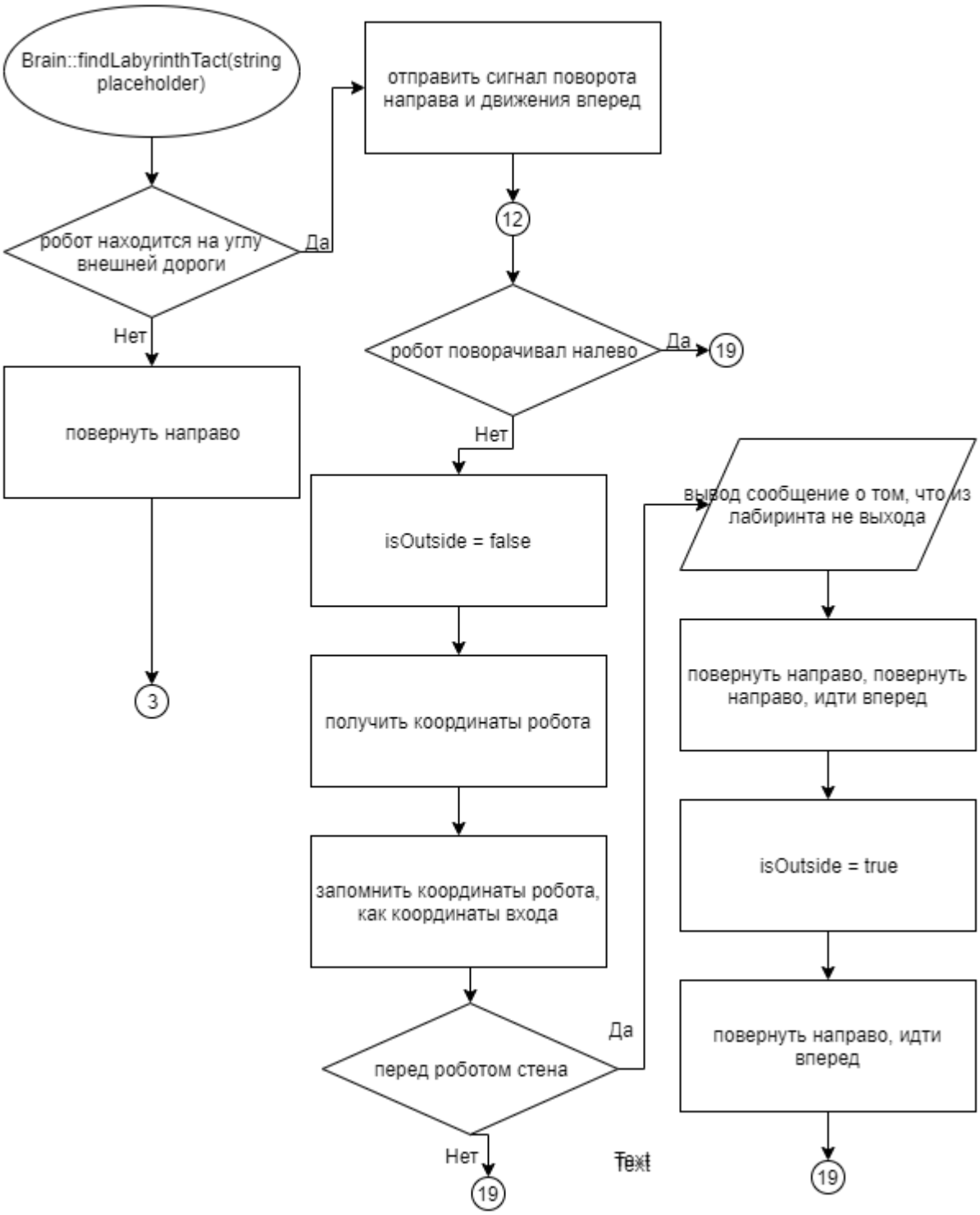

Рисунок 21 – Блок-схема алгоритма

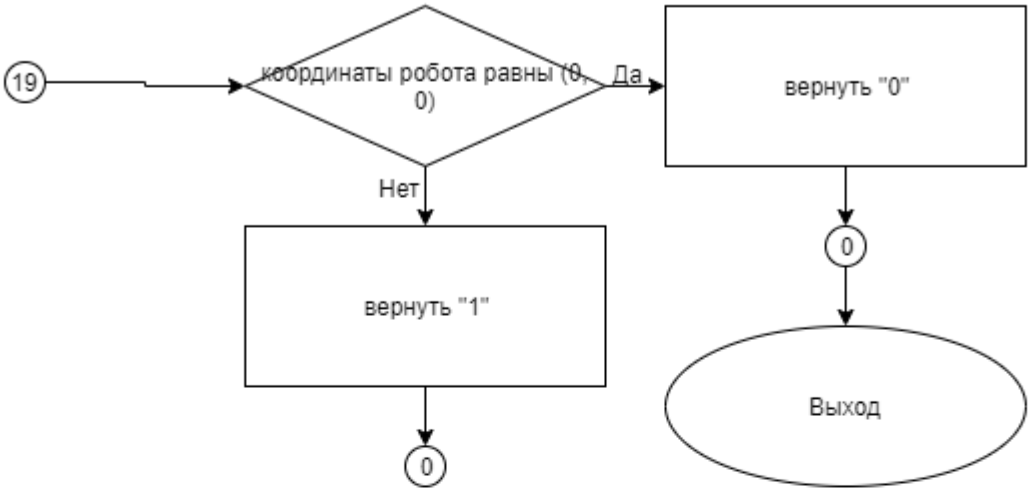

Рисунок 32 – Блок-схема алгоритма

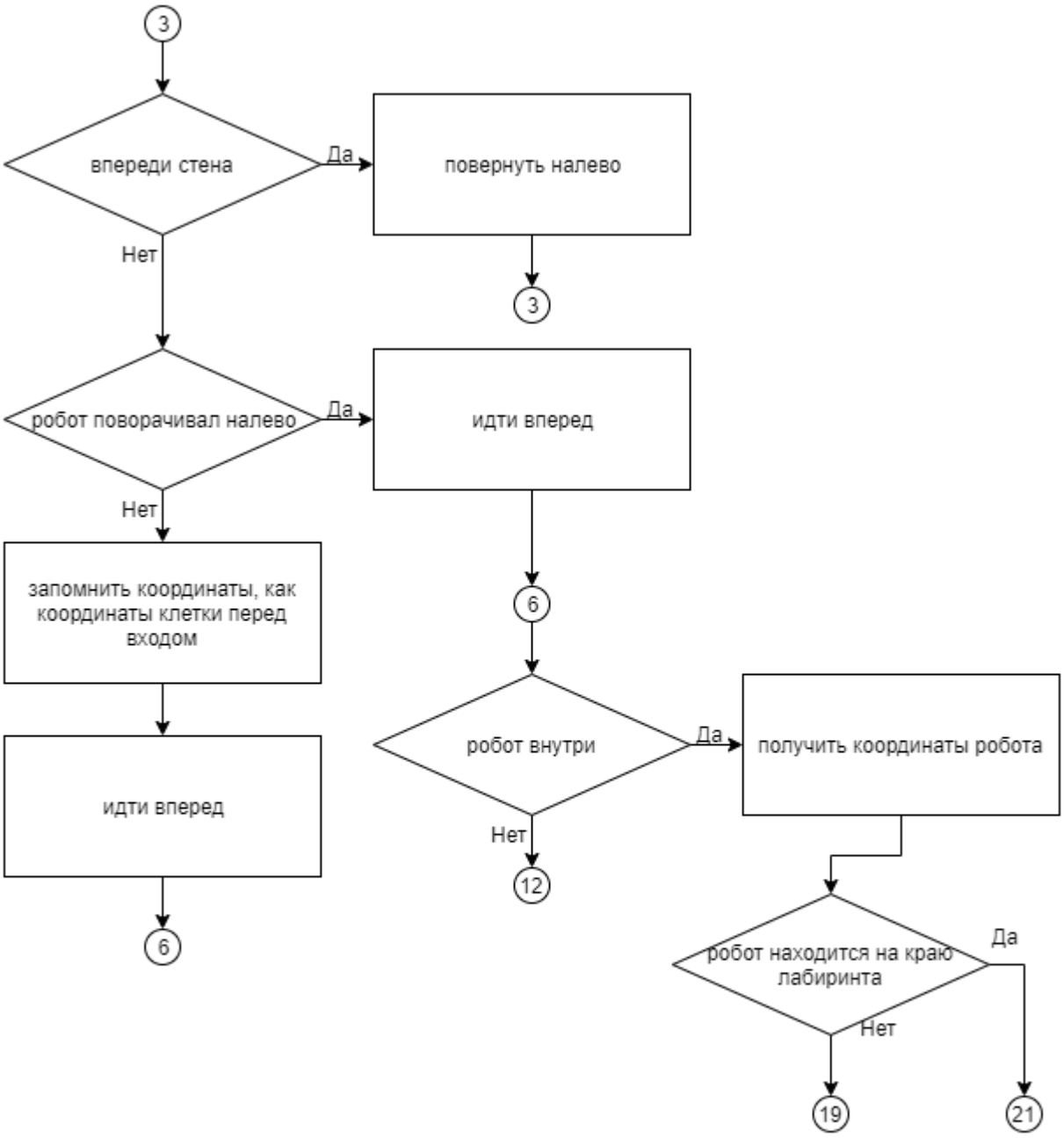

Рисунок 43 – Блок-схема алгоритма

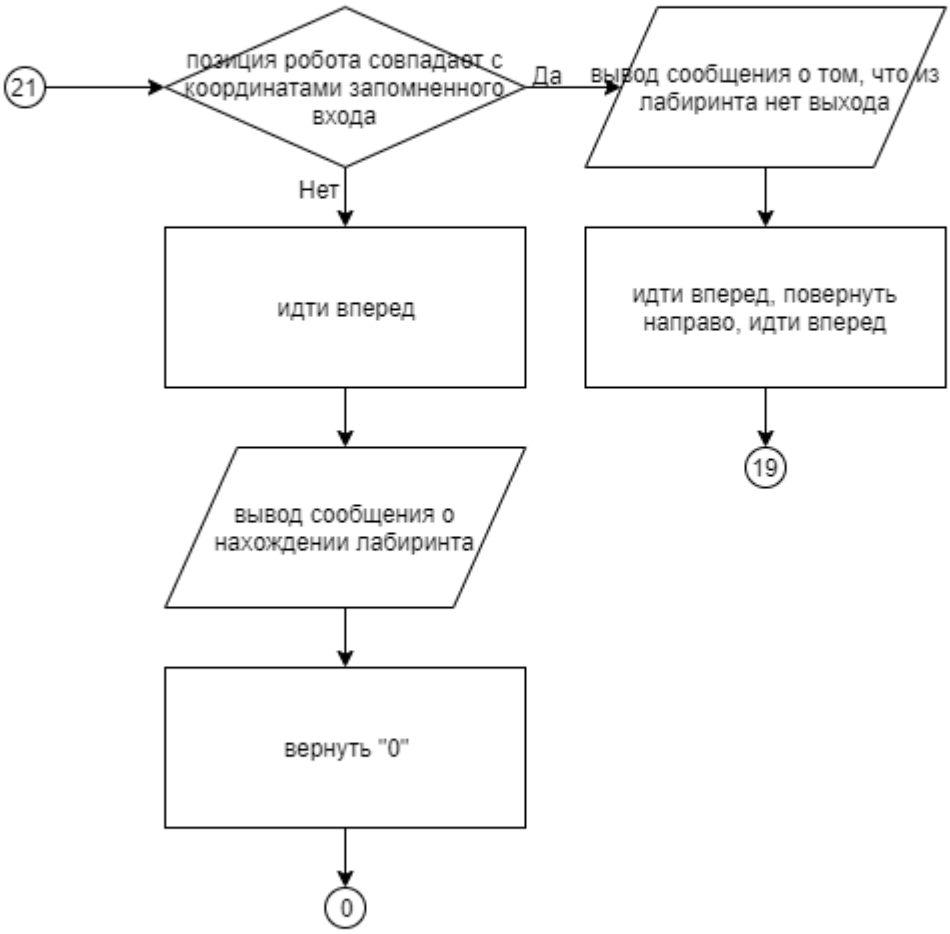

Рисунок 54 – Блок-схема алгоритма

5 КОД ПРОГРАММЫ

Программная реализация алгоритмов для решения задачи представлена ниже.

5.0 Файл Application.cpp

Листинг 1 – Application.cpp

```
#include "Application.h"
#include "Reader.h"
#include "Brain.h"
#include "Field.h"
#include "Bot.h"
#include "Printer.h"

void Application::errorOut(string headPath)
{
	showTree();
	cout << endl << "The head object " << headPath << " is not found";
}

bool Application::buildTree()
{
	this->setState(1);
	Reader* reader = new Reader(this, "reader");
	reader->setState(1);
	Brain* brain = new Brain(this, "brain");
	brain->setState(1);
	Field* field = new Field(this, "field");
	field->setState(1);
	Bot* bot = new Bot(this, "bot");
	bot->setState(1);
	Printer* printer = new Printer(this, "printer");
	printer->setState(1);
	//setStaet just in case, since we don't have actual user in our programm, we
control everything

	reader->set_connection(SIGNAL_D(Reader::insert),		(Base*)field,
HANDLER_D(Field::insertHandler)); //the insertion signal

	bot->set_connection(SIGNAL_D(Bot::getFieldStateSignal),	(Base*)field,
HANDLER_D(Field::getFieldStateHandler)); //bot getting state of specific field
//use emit_FSSignal
	bot->set_connection(SIGNAL_D(Bot::getFrontStateSignal),	(Base*)field,
HANDLER_D(Field::getFieldStateHandler)); //bot getting state of the field in front
of it //use emit_FSSignal

	brain->set_connection(SIGNAL_D(Brain::checkForwardSignal),	(Base*)bot,
HANDLER_D(Bot::checkForwardHandler));	//command from brain to return what's
```

```
infront
      brain->set_connection(SIGNAL_D(Brain::moveForwardSignal),        (Base*)bot,
HANDLER_D(Bot::moveForwardHandler)); //command from brain to move forward
      brain->set_connection(SIGNAL_D(Brain::turnSignal),               (Base*)bot,
HANDLER_D(Bot::turnHandler)); //command from brain to turn, the turn direction
should be selected via parameter
      brain->set_connection(SIGNAL_D(Brain::getCoordsSignal),          (Base*)bot,
HANDLER_D(Bot::getCoordsHandler)); //command to get current bot coordinates

      brain->set_connection(SIGNAL_D(Brain::printSignal),          (Base*)printer,
HANDLER_D(Printer::printHandler)); // command from brain to print something

      this->set_connection(SIGNAL_D(Application::tactSignal),        (Base*)brain,
HANDLER_D(Brain::findLabyrinthTact));//tact signals from application

      return true;

      /*//4-th task
      string headName;
      cin >> headName;
      this->setName(headName);
      cin >> headName;
      while(headName != "endtree")
      {
            string subName;
            cin >> subName;
            int classNum;
            cin >> classNum;
            //need to remove check for unique as names are no longer unique
            Base* tmpHead = this->findByPath(headName); //tmpHead nullptr when not
found
            if(tmpHead)
            {
                  switch(classNum) //choosing class
                  {
                        case 2:
                              new Reader(tmpHead, subName);
                              break;
                        case 3:
                              new Brain(tmpHead, subName);
                              break;
                        case 4:
                              new Field(tmpHead, subName);
                              break;
                        case 5:
                              new Bot(tmpHead, subName);
                              break;
                        case 6:
                              new Printer(tmpHead, subName);
                              break;
                  }
            }
            else
            {
                  errorOut(headName);
                  return false;
            }
```

```
            cin >> headName;
      }     //now connections (headName -> sender; subName -> receiver)
      //tmpHead-
>emit_signal(SIGNAL_D(remove_pointer<decltype(tmpHead)>::type::signal),  message);
//this is the closest I got to getting rid of switch casing. We use the type of
tmpHead (the problem is that tmpHead is a pointer to a Base class, even if it's
pointing to a Child, because we store Base*)
      cin >> headName;
      while(headName != "end_of_connections"){
            string subName;
            cin >> subName;
            Base* tmpHead = this->findByPath(headName);
            Base* tmpSub = this->findByPath(subName);
            if(!tmpHead) cout << "Object " << headName << " not found" << endl;
            else if(!tmpSub) cout << "Handler object " << subName << " not found"
<< endl;
            else{
                  tmpHead->set_connection(tmpHead->getSignal(),   tmpSub,   tmpSub-
>getHandler());
            }
            cin >> headName;
      }
      return true;*/
}

void Application::startApp()
{
      Base* reader = this->findByPath("/reader");
      Base* field = this->findByPath("/field");

      int insertStatus = reader->emit_FSSignal(SIGNAL_D(Reader::insert));
      if(!insertStatus) return; // we stop programm if insertStatus is 0 meaning
that reader got SHOWTREE

      Base* brain = this->findByPath("/brain");
      int INeedMoreTime = this->emit_FSSignal(SIGNAL_D(Application::tactSignal));
      while(INeedMoreTime)            INeedMoreTime            =            this-
>emit_FSSignal(SIGNAL_D(Application::tactSignal));

      /*showTree();
      this->setStateForAll(1); //turning all objects on
      string command;
      cin >> command;
      while(command != "END"){
            if(command == "EMIT"){
                  string path, text;
                  cin >> path;
                  getline(cin, text);
                  //text = text.substr(1); //getting rid of the first space
                  Base* tmpHead = this->findByPath(path);
                  if(!tmpHead) cout << endl << "Object " << path << " not found";
                  else tmpHead->emit_signal(tmpHead->getSignal(), text);
            }
            else if(command == "SET_CONNECT"){
                  string path1, path2;
                  cin >> path1 >> path2;
```

```
                  Base* tmpHead = this->findByPath(path1);
                  Base* tmpSub = this->findByPath(path2);
                  if(!tmpHead) cout << endl << "Object " << path1 << " not found";
                  else if(!tmpSub) cout << endl << "Handler object " << path2 << "
not found";
                  else    tmpHead->set_connection(tmpHead->getSignal(),    tmpSub,
tmpSub->getHandler());
            }
            else if(command == "DELETE_CONNECT"){
                  string path1, path2;
                  cin >> path1 >> path2;
                  Base* tmpHead = this->findByPath(path1);
                  Base* tmpSub = this->findByPath(path2);
                  if(!tmpHead) cout << endl << "Object " << path1 << " not found";
                  else if(!tmpSub) cout << endl << "Handler object " << path2 << "
not found";
                  else   tmpHead->delete_connection(tmpHead->getSignal(),   tmpSub,
tmpSub->getHandler());
            }
            else if(command == "SET_CONDITION"){
                  string path;
                  int state;
                  cin >> path >> state;
                  Base* tmpHead = this->findByPath(path);
                  if(!tmpHead) cout << endl << "Object " << path << " not found";
                  else tmpHead->setState(state);
            }
            //else cout << "Unknown command: " << command << endl;

            cin >> command;
      }*/
}
```

5.1 Файл Application.h

Листинг 2 – Application.h

```
#ifndef __APPLICATION_H__
#define __APPLICATION_H__
#include "Base.h"
#include "Reader.h"
#include "Brain.h"
#include "Field.h"
#include "Bot.h"
#include "Printer.h"

class Application : public Base
{
private:
      //vector<shared_ptr<Base>> allSubObj;
public:
      Application(Base*  head,  string  name  =  "Base_object")  :  Base(head,  name)
{childType = 1;}
```

```
	bool buildTree();
	void startApp();
	void errorOut(string headPath);
	/*void signal(string& mes){
		mes += " (class: 1)";
		cout << endl << "Signal from " << this->getPath();
	}
	void handler(string mes){cout << endl << "Signal to " << this->getPath() <<
" Text: " << mes;}*/
	void tactSignal(string& placeholder){}
};

#endif
```

5.2 Файл Base.cpp

Листинг 3 – Base.cpp

```
#include "Base.h"

Base::Base(Base* head, string name)
{
	Base* tmp = head;
	if(head) while(tmp->head) tmp = tmp->head; // finding root
	//if(head && tmp->findInSubTree(name)) delete(this); //not creating object
if name is occupied

	this->name = name;
	state = 0;
	this->head = head;
	if(head != nullptr)
	{
		head->subordinates.push_back(this);
	}
}

Base::~Base() //if commented then we're using shared ptr and this will be done
automaticly
{
	//cout << name << " " << subordinates.size() << endl;
	//if(0 < subordinates.size()) cout << "a" << endl;
	//if(0 < subordinates.size()) delete subordinates[0];
	for(int i = 0; i < outcomes.size(); i++)
	{
		delete(outcomes[i]);
	}
	for(int i = 0; i < subordinates.size(); i++)
	{
		delete(subordinates[i]);
	}
}

bool Base::setName(string name)
```

```
{
	Base* tmp = this;
	while(tmp->head) tmp = tmp->head; // finding root
	if(!(tmp->findInSubTree(name))) //no object with such name found (we can use
it)
	{
		this->name = name;
		return true;
	}
	return false;
	//this->name = name;
}

string Base::getName()
{
	return name;
}

void Base::setState(int state)
{
	if(this->head)
	{
		if(this->head->state)
		{
			this->state = state;
			if(state == 0) for(int i = 0; i < subordinates.size(); i++)
subordinates[i]->setState(0);
		}
		else{//head->state = 0
			this->state = 0;
			for(int i = 0; i < subordinates.size(); i++) subordinates[i]-
>setState(0);
		}
	}
	else
	{
		this->state = state;
		if(state == 0) for(int i = 0; i < subordinates.size(); i++)
subordinates[i]->setState(0);
	}
}

void Base::setStateForAll(int state){ //setting starts from this, so if this->head
has state 0, then nothing will happen
	//if(this->head && this->head->state == 0) return;
	this->setState(state);
	for(int i = 0; i < subordinates.size(); i++){
		subordinates[i]->setStateForAll(state);
	}
}

int Base::getState()
{
	return state;
}

void Base::setHead(Base* head)
```

```
{
	this->head->subordinates.erase(find(subordinates.begin(),
subordinates.end(), this)); //deleting "this" from previous head
	this->head = head;
	head->subordinates.push_back(this);
}

Base* Base::getHead()
{
	return this->head;
}

Base* Base::getSubordinateByName(string subName)
{
	for(int i = 0; i < this->subordinates.size(); i++)
	{
		if(subordinates[i]->name == subName) return subordinates[i];
	}
	return nullptr;
}

void Base::showTree(int depth)
{
	if(this->head) cout << endl;
	else cout << "Object tree" << endl;
	for(int i = 0; i < depth; i++) cout << "    ";
	cout << name;
	for(int i = 0; i < subordinates.size(); i++) subordinates[i]->showTree(depth
+ 1);
}

void Base::showTreeState(int depth)
{
	if(this->head) cout << endl;
	for(int i = 0; i < depth; i++) cout << "    ";
	if(state) cout << name << " is ready";
	else cout << name << " is not ready";
	for(int   i   =   0;   i   <   subordinates.size();   i++)   subordinates[i]-
>showTreeState(depth + 1);
}

Base* Base::findInSubTree(string name)
{
	if(this->name == name) return(this);
	Base* searched;
	for(int i = 0; i < this->subordinates.size(); i++)
	{
		searched = subordinates[i]->findInSubTree(name);
		if(searched) return(searched);
	}
	return(nullptr);
}

Base* Base::findByName(string name)
{
	Base* tmp = this;
	while(tmp->head) tmp = tmp->head; // finding root
```

```
	return(tmp->findInSubTree(name));
}

Base* Base::findByPath(string path)
{
	if(!this || path.empty()) return(nullptr);
	if(path.substr(0, 2) == "//"){
		return findByName(path.substr(2));
	}
	if(path.substr(0, 1) == "/"){ // / or /sth or /sth/sth
		//cout << path << endl;
		int nextSlash = path.find("/", 1);
		Base* tmpRoot = this;
		while(tmpRoot->head) tmpRoot = tmpRoot->head; // finding root
		//cout << nextSlash << endl;
		if(nextSlash == -1 && path.size() == 1) return tmpRoot; //root
		if(nextSlash == -1) return(tmpRoot->getSubordinateByName(path.substr(1))); // one after root
		Base* tmp = tmpRoot->getSubordinateByName(path.substr(1, nextSlash - 1)); //more than one after root
		return(tmp->findByPath(path.substr(nextSlash + 1))); //if tmp = nullptr then findByPath returns nullptr
	}
	else if(path.substr(0, 1) == "."){
		if(path.size() == 1) return(this); // just .
		return(this->findByPath(path.substr(2))); // ./sth
	}
	else{
		//cout << path << endl;
		int nextSlash = path.find("/");
		if(nextSlash == -1) return(this->getSubordinateByName(path)); // one after our element
		Base* tmp = this->getSubordinateByName(path.substr(0, nextSlash)); //more than one after our element
		return(tmp->findByPath(path.substr(nextSlash + 1)));
	}
}

string Base::getPath()
{
	string path = "";
	Base* tmp = this;
	if(!tmp->head) return "/";
	while(tmp->head){ // there is a head
		path = "/" + tmp->getName() + path;
		tmp = tmp->head;
	}
	return path;
}

void Base::set_connection(TYPE_SIGNAL signal, Base* destination, TYPE_HANDLER handler)
{
	if(!this || !destination) return;
	outcome* out = new outcome(destination, signal, handler);
	//income* in = new income(this, handler);
	for(int i = 0; i < outcomes.size(); i++){
```

```
            if(*(outcomes[i]) == *out){
                  //cout << "test_equal"; //this works
                  delete out; //we don't push back here, so we need to destroy
                  //delete in;
                  return; //no equal connections
            }
      }
      this->outcomes.push_back(out); //new outcome and income will bedestroyed by
class destructor
      //this->incomes.push_back(in);
}

void Base::delete_connection(TYPE_SIGNAL signal, Base* destination, TYPE_HANDLER
handler)
{
      //cout << "delete";
      if(!this || !destination) return;
      outcome* out = new outcome(destination, signal, handler);
      //income* in = new income(this, handler);
      {//blocked them to deal with error of "it" not changing type properly
            int ind = -1;
            for(int i = 0; i < outcomes.size(); i++){
                  if(*(outcomes[i]) == *out) ind = i;
            }
            //auto it = find(outcomes.begin(), outcomes.end(), out); //does not
work, dunno why
            if(ind != -1){
                  delete outcomes[ind]; //don't know wheter only one or both
delete and erase
                  this->outcomes.erase(outcomes.begin() + ind); //potential memory
leak if left alone
                  //cout << "test_delete_out";
            }
            //check whether the overloaded operator == is working
      }
      //delete out;
      //delete in;
}

void Base::emit_signal(TYPE_SIGNAL signal, string mes)
{
      if(!this || state == 0){
            //cout << "not emitting ";
            return; //if this is nullptr or obj is not active we do nothing
      }
      (this->*signal)(mes);
      for(int i = 0; i < outcomes.size(); i++){
            if(outcomes[i]->signal == signal){
                  Base* destination = outcomes[i]->receiver;
                  if(destination->getState() != 0){
                        (destination->*(outcomes[i]->handler))(mes);
                  }

            }
      }
}
```

```
int Base::emit_FSSignal(TYPE_SIGNAL signal, string posI100J) //default coordinates
will be erased by front signal, otherwise default is (0,0)
{
	//-2 means something is turned off or there is no such connection
	if(!this || state == 0){
		return -2;
	}
	(this->*signal)(posI100J);  //if  it's  a  placeholder  signal  then  nothing
happens, otherwise we can decide the coordinate in signal method
	for(int i = 0; i < outcomes.size(); i++){
		if(outcomes[i]->signal == signal){
			Base* destination = outcomes[i]->receiver;
			if(destination->getState() != 0){
				return                    stoi((destination->*(outcomes[i]-
>handler))(posI100J));
			}
		}
	}
	return -2;
}

/*TYPE_SIGNAL Base::getSignal(){
	switch(this->getChildType()){
		case 1: return SIGNAL_D(Application::signal); //we return => we don't
need a break
		case 2: return SIGNAL_D(Reader::signal);
		case 3: return SIGNAL_D(Brain::signal);
		case 4: return SIGNAL_D(Field::signal);
		case 5: return SIGNAL_D(Bot::signal);
		case 6: return SIGNAL_D(Printer::signal);
		default: return nullptr;
	}
}

TYPE_HANDLER Base::getHandler(){
	switch(this->getChildType()){
		case 1: return HANDLER_D(Application::handler);
		case 2: return HANDLER_D(Reader::handler);
		case 3: return HANDLER_D(Brain::handler);
		case 4: return HANDLER_D(Field::handler);
		case 5: return HANDLER_D(Bot::handler);
		case 6: return HANDLER_D(Printer::handler);
		default: return nullptr;
	}
}*/
```

5.3 Файл Base.h

Листинг 4 – Base.h

```
#ifndef __BASE_H__
#define __BASE_H__
```

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <memory>
#include <typeinfo>
#include <type_traits>

using namespace std;

class Base;

typedef void(Base::*TYPE_SIGNAL)(string&);
typedef string(Base::*TYPE_HANDLER)(string);
#define SIGNAL_D(signal_f)(TYPE_SIGNAL)(&signal_f)
#define HANDLER_D(handler_f)(TYPE_HANDLER)(&handler_f)

class Base
{
private:
	string name;
	Base* head;
	int state;
	vector <Base*> subordinates;


	struct outcome{ //what goes from this object
		Base* receiver;
		TYPE_SIGNAL signal;
		TYPE_HANDLER handler;
		outcome(Base*	receiver,	TYPE_SIGNAL	signal,	TYPE_HANDLER
handler):receiver(receiver), signal(signal), handler(handler){};
		bool operator==(const outcome& other) const{
			return(receiver==other.receiver	&&	signal==other.signal	&&
handler==other.handler);
		}
	};
	/*struct income{ //what comes to this object
		Base* sender;
		TYPE_HANDLER handler;
		income(Base*	sender,	TYPE_HANDLER	handler):sender(sender),
handler(handler){};
		bool operator==(const income& other) const{
			return(sender==other.sender && handler==other.handler);
		}
	};*/
	vector<outcome*> outcomes;
	//vector<income*> incomes;
protected:
	int childType;
public:
	Base(Base* head, string name = "Base_object");
	~Base();
	bool setName(string name);
	string getName();
	void setState(int state);
	void setStateForAll(int state);
```

```
	int getState();
	void setHead(Base* head);
	int getChildType(){return childType;}
	Base* getHead();
	Base* getSubordinateByName(string subName);
	Base* findInSubTree(string name);
	Base* findByName(string name);
	void showTree(int depth = 0);
	void showTreeState(int depth = 0);
	Base* findByPath(string path);
	string getPath();
	void set_connection(TYPE_SIGNAL signal, Base* destination, TYPE_HANDLER
handler);
	void delete_connection(TYPE_SIGNAL signal, Base* destination, TYPE_HANDLER
handler);
	void emit_signal(TYPE_SIGNAL signal, string mes = "foo");
	int emit_FSSignal(TYPE_SIGNAL signal, string posI100J = "0");
	/*TYPE_SIGNAL getSignal();
	TYPE_HANDLER getHandler();*/
};

#include "Application.h"
#include "Reader.h"
#include "Brain.h"
#include "Field.h"
#include "Bot.h"
#include "Printer.h"

#endif
```

5.4 Файл Bot.cpp

Листинг 5 – Bot.cpp

```
#include "Bot.h"

using namespace std;

void Bot::absoluteMove(char direction)
{
	//direction can be 'u' - up; 'r' - right; 'd' - down; 'l' - left
	int newPosi;
	int newPosj;
	int posState = -2;
	switch(direction)
	{
		case 'u':
			//move up, check for walls and end of field
			newPosi = posi-1;
			newPosj = posj;
			break;
		case 'r':
			//move right, check for walls and end of field
```

```
                  newPosi = posi;
                  newPosj = posj+1;
                  break;
            case 'd':
                  //move down, check for walls and end of field
                  newPosi = posi+1;
                  newPosj = posj;
                  break;
            case 'l':
                  //move left, check for walls and end of field
                  newPosi = posi;
                  newPosj = posj-1;
                  break;
            default:
                  break;
      }
      posState      =      this->emit_FSSignal(SIGNAL_D(Bot::getFieldStateSignal),
to_string(newPosi*100 + newPosj)); //1 - wall; 0 - space; -1 - out of bounds; -2 -
error
      if(posState == 0){ //if we can we move
            posi = newPosi;
            posj = newPosj;
      }
}

void Bot::getFrontStateSignal(string& posI100J){
      int newPosi;
      int newPosj;
      switch(facing){
            case 'N':
                  newPosi = posi-1;
                  newPosj = posj;
                  break;
            case 'E':
                  newPosi = posi;
                  newPosj = posj+1;
                  break;
            case 'S':
                  newPosi = posi+1;
                  newPosj = posj;
                  break;
            case 'W':
                  newPosi = posi;
                  newPosj = posj-1;
                  break;
            default:
                  cout << "facing error";
                  break;
      }
      posI100J = to_string(newPosi*100 + newPosj);
}

string Bot::checkForwardHandler(string placeholder){
      string            out            =            to_string(            this-
>emit_FSSignal(SIGNAL_D(Bot::getFrontStateSignal)) );
      return out;
}
```

```
string Bot::moveForwardHandler(string placeholder){
	int newPosi;
	int newPosj;
	int posState = -2;
	switch(facing)
	{
		case 'N':
			//move up, check for walls and end of field
			newPosi = posi-1;
			newPosj = posj;
			break;
		case 'E':
			//move right, check for walls and end of field
			newPosi = posi;
			newPosj = posj+1;
			break;
		case 'S':
			//move down, check for walls and end of field
			newPosi = posi+1;
			newPosj = posj;
			break;
		case 'W':
			//move left, check for walls and end of field
			newPosi = posi;
			newPosj = posj-1;
			break;
		default:
			break;
	}
	posState	=	this->emit_FSSignal(SIGNAL_D(Bot::getFieldStateSignal),
to_string(newPosi*100 + newPosj)); //1 - wall; 0 - space; -1 - out of bounds; -2 -
error
	if(posState == 0){ //if we can we move
		posi = newPosi;
		posj = newPosj;
	}
	return "";
}

string Bot::turnHandler(string direction){
	//direction is "R" or "L"
	string directions = "NESWNESW";
	int faceInd = directions.find(facing);
	if(direction == "R") ++faceInd;
	if(direction == "L") faceInd += 4-1; //just -1 will throw us out of bounds
on N
	facing = directions[faceInd];
	return "";
}

string Bot::getCoordsHandler(string placeholder){
	return(to_string(100*posi + posj));
}
```

5.5 Файл Bot.h

Листинг 6 – Bot.h

```
#ifndef __CHILD5_H__
#define __CHILD5_H__
#include "Base.h"

class Bot : public Base
{
private:
	int posi = 0;
	int posj = 1;
	char facing = 'E'; //N; E; S; W;
public:
	Bot(Base* head, string name = "Base_object") : Base(head, name) {childType =
5; posi = 0; posj = 1; facing = 'E';}
	/*void signal(string& mes){
		mes += " (class: 5)";
		cout << endl << "Signal from " << this->getPath();
	}
	void handler(string mes){cout << endl << "Signal to " << this->getPath() <<
" Text: " << mes;}*/
	void getFieldStateSignal(string& placeholder){}
	void getFrontStateSignal(string& posI100J); //signal creates position by
looking at facing variable

	string checkForwardHandler(string placeholder);
	string moveForwardHandler(string placeholder);
	string turnHandler(string direction); //diretion is "R" - for right; "L" -
for left
	string getCoordsHandler(string placeholder);

	int getPosi(){return posi;}
	void setPosi(int posi){this->posi = posi;}
	int getPosj(){return posj;}
	void setPosj(int posj){this->posj = posj;}
	char getFacing(){return facing;}
	void setFacing(char facing){this->facing = facing;}

	void absoluteMove(char direction);
};

#endif
```

5.6 Файл Brain.cpp

Листинг 7 – Brain.cpp

```
#include "Brain.h"
```

```
using namespace std;

string Brain::findLabyrinthTact(string placeholder){ //it's a handler
	/*what we need:
		checkFront  done:  emit_FSSignal(SIGNAL_D(Brain::checkForwardSignal));
//1 - wall; 0 - space; -1 - out of bounds; -2 - error (there should be none)
		moveForward done: emit_signal(SIGNAL_D(Brain::moveForwardSignal));
		turnLeft done: emit_signal(SIGNAL_D(Brain::turnSignal), "L");
		turnRight done: emit_signal(SIGNAL_D(Brain::turnSignal), "R");
		print  done:  emit_signal(SIGNAL_D(Brain::printSignal),  "whatever  we
need to print");
		get   coords   done:   emit_FSSignal(SIGNAL_D(Brain::getCoordsSignal));
//return posI100J
	*/

	if(isOutside  &&  emit_FSSignal(SIGNAL_D(Brain::checkForwardSignal))  ==  -1){
//we are at the corner and the next turn is not a labyrinth entry
		emit_signal(SIGNAL_D(Brain::turnSignal), "R");
		emit_signal(SIGNAL_D(Brain::moveForwardSignal));
	}
	else{
		emit_signal(SIGNAL_D(Brain::turnSignal),  "R");  //turn  right,  try  to
go, if can't then spin left
		bool turnedRight = true;
		while(emit_FSSignal(SIGNAL_D(Brain::checkForwardSignal)) != 0){
			emit_signal(SIGNAL_D(Brain::turnSignal), "L"); //while we can't
move forward we turn left
			turnedRight = false;
		}
		if(isOutside && turnedRight){ //we have to remember the cell before
the entry to print it, because the task wants us to, but only if on the outside
road we turned right (no turns left during the spin)
			int                              entryCoords                              =
emit_FSSignal(SIGNAL_D(Brain::getCoordsSignal));
			entryPrintPosi = entryCoords / 100;
			entryPrintPosj = entryCoords % 100;
		}

		emit_signal(SIGNAL_D(Brain::moveForwardSignal));


		if(!isOutside){ //we were and are inside => have to check wheather
we've found exit
			int curCoords = emit_FSSignal(SIGNAL_D(Brain::getCoordsSignal));
			int posi = curCoords / 100;
			int posj = curCoords % 100;
			if(posi  ==  1  ||  posi  ==  22-2  ||  posj  ==  1  ||  posj  ==  22-2){
//it's an exit (22 is metainfo => size of field)
				if(posi == entryPosi && posj == entryPosj){ //exit = entry
					//There is no way out of the maze (1, 3)
					if(!isFirstOutput)
emit_signal(SIGNAL_D(Brain::printSignal), "\n");
					emit_signal(SIGNAL_D(Brain::printSignal),  "There  is
no  way  out  of  the  maze  ("  +  to_string(entryPrintPosi  +  1)  +  ",  "  +
to_string(entryPrintPosj + 1) + ")");
					isFirstOutput = false;
					emit_signal(SIGNAL_D(Brain::moveForwardSignal));
```

```
//get out of the maze to the outside road
                              isOutside = true;
                              emit_signal(SIGNAL_D(Brain::turnSignal), "R");
                              emit_signal(SIGNAL_D(Brain::moveForwardSignal));
//move away from already checked entry
                        }
                        else{ //exit != entry
                              //we'll have to move one step forward because the
task thinks that the exit of the maze is the cell on the outside road
                              emit_signal(SIGNAL_D(Brain::moveForwardSignal));

                              curCoords                                          =
emit_FSSignal(SIGNAL_D(Brain::getCoordsSignal));
                              posi = curCoords / 100;
                              posj = curCoords % 100;

                              //Maze (14, 22) (12, 1)
                              if(!isFirstOutput)
emit_signal(SIGNAL_D(Brain::printSignal), "\n");
                              emit_signal(SIGNAL_D(Brain::printSignal), "Maze (" +
to_string(entryPrintPosi + 1) + ", " + to_string(entryPrintPosj + 1) + ") (" +
to_string(posi + 1) + ", " + to_string(posj + 1) + ")");
                              isFirstOutput = false;
                              return("0"); //we found a maze, we finish
                        }
                  }
            }
            else if(turnedRight){//we are outside and we only turned right => we
found entry to labyrinth => we have to remember it (if didn't turn just right =>
we proceed with outside road => we don't need to do anything)
                  isOutside = false;
                  int                          entryCoords                       =
emit_FSSignal(SIGNAL_D(Brain::getCoordsSignal));
                  entryPosi = entryCoords / 100;
                  entryPosj = entryCoords % 100;
                  if(emit_FSSignal(SIGNAL_D(Brain::checkForwardSignal))   !=   0){
//that's a one cell maze
                        if(!isFirstOutput)
emit_signal(SIGNAL_D(Brain::printSignal), "\n");
                        emit_signal(SIGNAL_D(Brain::printSignal), "There is no way
out   of   the   maze   ("   +   to_string(entryPrintPosi   +   1)   +   ",   "   +
to_string(entryPrintPosj + 1) + ")");
                        isFirstOutput = false;
                        emit_signal(SIGNAL_D(Brain::turnSignal), "R");
                        emit_signal(SIGNAL_D(Brain::turnSignal), "R");
                        emit_signal(SIGNAL_D(Brain::moveForwardSignal)); //get out
of the maze to the outside road
                        isOutside = true;
                        emit_signal(SIGNAL_D(Brain::turnSignal), "R");
                        emit_signal(SIGNAL_D(Brain::moveForwardSignal));    //move
away from already checked entry
                  }
            }
      }
      if(emit_FSSignal(SIGNAL_D(Brain::getCoordsSignal))  ==  0)  return  "0";  //0
means 0*100 + 0 => (0,0)
      return "1"; //meaning we still have work, and we need more time ticks from
```

```
application (0 is returned after we find a labytinth or we come to (0,0))
}
```

5.7 Файл Brain.h

Листинг 8 – Brain.h

```
#ifndef __CHILD3_H__
#define __CHILD3_H__
#include "Base.h"

class Brain : public Base
{
private:
	int entryPosi = -1;
	int entryPosj = -1;
	int entryPrintPosi = -1;
	int entryPrintPosj = -1;
	bool isOutside = true;
	bool isFirstOutput = true;
public:
	Brain(Base* head, string name = "Base_object") : Base(head, name) {childType
= 3;}
	/*void signal(string& mes){
		mes += " (class: 3)";
		cout << endl << "Signal from " << this->getPath();
	}
	void handler(string mes){cout << endl << "Signal to " << this->getPath() <<
" Text: " << mes;}*/
	void checkForwardSignal(string& placeholder){}
	void moveForwardSignal(string& placeholder){}
	void turnSignal(string& placeholder){}
	void printSignal(string& placeholder){}
	void getCoordsSignal(string& placeholder){}

	string findLabyrinthTact(string placeholder); //this is actually the main
thing which performs the task
};

#endif
```

5.8 Файл Field.cpp

Листинг 9 – Field.cpp

```
#include "Field.h"

using namespace std;
```

```
string Field::insertHandler(string fieldStr)
{
	// as we push_back, we can use this handler only once in programm, since
there only one field, it's fine
	if(fieldStr == "SHOWTREE") return "0"; //means we have to stop the programm
	int lineLength = fieldStr.find("\n");
	for(int j = 0, i = 0; i*(lineLength + 1) + j < fieldStr.size(); j++){
		if(fieldStr[i*(lineLength + 1) + j] == '\n'){
			++i; // we go to the next line
			j = -1; //it'll become 0 on next iteration
		}
		else{
			if(j == 0) field.push_back(vector<int>()); //if the line is new
we add a new line in field
			field[i].push_back((int)(fieldStr[i*(lineLength + 1) + j] -
'0'));
		}
	}
	return "1"; //means everything's fine
	//need to uses emit_FSSignal(SIGNAL_D(Reader::insert)) and it'll return 0 or
1
}

string Field::getFieldStateHandler(string posI100J)
{
	int i = stoi(posI100J) / 100;
	int j = stoi(posI100J) % 100;
	if(i < 0 || i >= field.size() || j < 0 || j >= field[i].size()) return
to_string(-1); //out of bounds
	return to_string(field[i][j]);
}
```

5.9 Файл Field.h

Листинг 20 – Field.h

```
#ifndef __CHILD4_H__
#define __CHILD4_H__
#include "Base.h"

class Field : public Base
{
private:
	vector<vector<int>> field;
public:
	Field(Base* head, string name = "Base_object") : Base(head, name) {childType
= 4;}
	/*void signal(string& mes){
		mes += " (class: 4)";
		cout << endl << "Signal from " << this->getPath();
	}
	void handler(string mes){cout << endl << "Signal to " << this->getPath() <<
" Text: " << mes;}*/
```

```
	string insertHandler(string fieldStr);
	string getFieldStateHandler(string posI100J);
};

#endif
```

5.10 Файл main.cpp

Листинг 31 – main.cpp

```
#include "Application.h"

int main()
{
	Application app(nullptr);
	if(app.buildTree()) app.startApp();
	return(0);
}
```

5.11 Файл Printer.h

Листинг 42 – Printer.h

```
#ifndef __CHILD6_H__
#define __CHILD6_H__
#include "Base.h"

class Printer : public Base
{
public:
	Printer(Base* head, string name = "Base_object") : Base(head, name)
{childType = 6;}
	/*void signal(string& mes){
		mes += " (class: 6)";
		cout << endl << "Signal from " << this->getPath();
	}
	void handler(string mes){cout << endl << "Signal to " << this->getPath() <<
" Text: " << mes;}*/
	string printHandler(string mes){
		cout << mes;
		return "";
	}
};

#endif
```

5.12 Файл Reader.cpp

Листинг 53 – Reader.cpp

```
#include "Reader.h"

using namespace std;

void Reader::insert(string& field)
{
	field = "";
	for(int i = 0; i < 22; i++){
		string tmp;
		cin >> tmp;
		if(tmp == "SHOWTREE"){
			Base* tmpRoot = this;
			while(tmpRoot->getHead()) tmpRoot = tmpRoot->getHead();
			tmpRoot->showTreeState();
			field =  "SHOWTREE"; //this means we have to end programm
			return;
			i--; //showtree does not count in the number of field lines
		}
		else{
			//tmp is a line of field
			field += tmp + "\n";
		}
	}
}
```

5.13 Файл Reader.h

Листинг 64 – Reader.h

```
#ifndef __CHILD2_H__
#define __CHILD2_H__
#include "Base.h"

class Reader : public Base
{
public:
	Reader(Base*  head,  string  name  =  "Base_object")  :  Base(head,  name)
{childType = 2;}
	/*void signal(string& mes){
		mes += " (class: 2)";
		cout << endl << "Signal from " << this->getPath();
	}
	void handler(string mes){cout << endl << "Signal to " << this->getPath() <<
" Text: " << mes;}*/
	void insert(string& field);
};
```

```
#endif
```

6 ТЕСТИРОВАНИЕ

Результат тестирования программы представлен в таблице **Error! Reference source not found.**.

Таблица 17 – Результат тестирования программы

Входные данные	Ожидаемые выходные данные	Фактические выходные данные
00000000000000000000 01011101111011111110 01000111000000011111110 01011111111111111100000 01000111111011000011110 01111111111000011111110 01111111111111111111110 01111111111111111111110 01110111111111111111110 01110111111111111110110 00000000001111111110110 01101111101111111110110 01101100000110000000000 01111111110000111110110 01111111111111110000110 01010111011111110111110 01010111011111110111110 00000000111111111111110 01111111111111111111110 01111111111111111111110 00000000000000000000000	There is no way out of the maze (1, 3) There is no way out of the maze (1, 7) There is no way out of the maze (1, 12) There is no way out of the maze (4, 22) Maze (14, 22) (12, 1)	There is no way out of the maze (1, 3) There is no way out of the maze (1, 7) There is no way out of the maze (1, 12) There is no way out of the maze (4, 22) Maze (14, 22) (12, 1)
00000000000000000000 01011111111011111111110 01000111000000011111110 01011111111111111100000 01000111111011000011110 01111111111000011111110 01111111111111111111110 01111111111111111111110 01110111111111111111110 01110111111111111110110 00000000001111111110110 01101111101111111110110 01101100000110000000000 01111111110000111110110 01111111111111110000110 01010111011111110111110 01010111011111110111110 00000000111111111111110 01111111111111111111110 01111111111111111111110 00000000000000000000000	There is no way out of the maze (1, 3) There is no way out of the maze (1, 12) There is no way out of the maze (4, 22) Maze (14, 22) (12, 1)	There is no way out of the maze (1, 3) There is no way out of the maze (1, 12) There is no way out of the maze (4, 22) Maze (14, 22) (12, 1)

ЗАКЛЮЧЕНИЕ

В процессе прохождения курса объектно-ориентированного программирования, мною были получены знания и навыки, касающиеся концепции объектно-ориентированного программирования. Также в процессе обширной практики был получен опыт программирования в стиле объектно-ориентирования программирования. Была освоена работа с классами и их объектами, создание связей между ними, наследование одних классов от других, прочие особенности объектно-ориентированного программирования и различные способы работы со всем вышеперечисленным. Также в процессе прохождения курса я был ознакомлен с идеологией объектно-ориентированного программирования и работал в соответствии с ней. По результатам курса было выполнено множество практических заданий, в том числе и данная курсовая работа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Васильев А.Н. Объектно-ориентированное программирование на C++. Издательство: Наука и Техника. Санкт-Петербург, 2016г. 543 стр.

2. Шилдт Г. C++: базовый курс. 3-е изд. Пер. с англ.. — М.: Вильямс, 2017. — 624 с.

3. Методическое пособие для проведения практических заданий, контрольных и курсовых работ по дисциплине «Объектно-ориентированное программирование» [Электронный ресурс] – URL: https://mirea.aco-avrora.ru/student/files/methodichescoe_posobie_dlya_laboratorny h_rabot_3.pdf (дата обращения 05.05.2021).

4. 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).

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).