МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Отчет по лабораторной работе №3
по дисциплине
«Теория автоматов»

Выполнил студент гр. ИВТб-2301-05-00	__________/Макаров С.А./
Преподаватель	__________/Мельцов В.Ю./

Киров 2025

Цель

Получить навыки разработки алгоритма, реализующего автомат, который реализовывает бота для игры «Морской бой».

Задание

Разработать алгоритм работы бота для игры «Морской бой», разработать программу, принять участие в турнире.

Решение

Макаров Станислав ИВТб-2301

Для игры используется четыре стратегии. Сперва используется стратегия для поиска 4-х палубного корабля. Когда он будет убит происходит переключение стратегии для поиска 3-х палубных. По аналогии происходит переключение на стратегии для поиска 2-х и 1-х палубных кораблей.

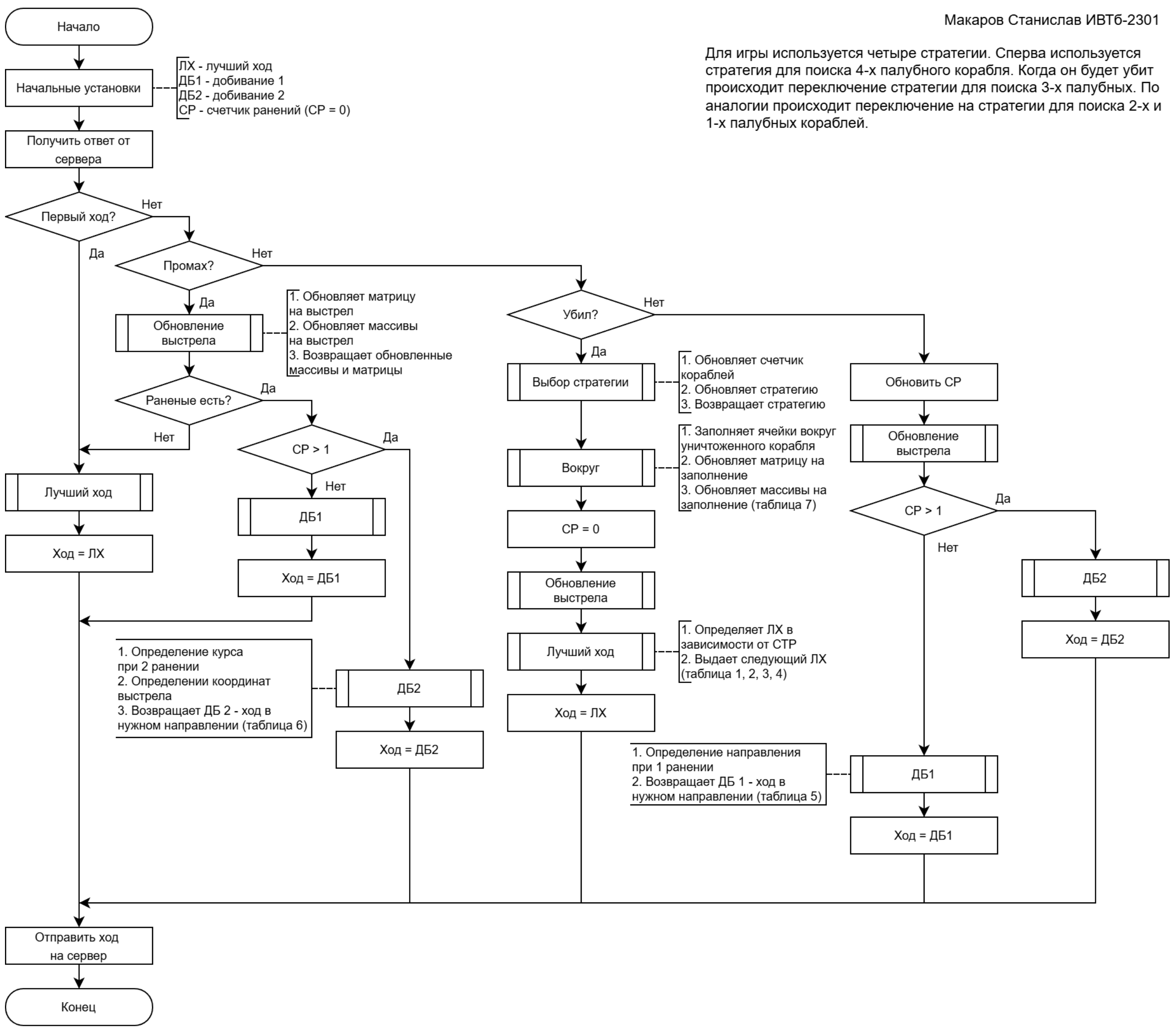

Начало
Начальные установки
ЛХ - лучший ход
ДБ1 - добивание 1
ДБ2 - добивание 2
СР - счетчик ранений (СР = 0)
Получить ответ от сервера
Первый ход?
Нет
Да
Промах?
Нет
Да
Обновление выстрела
1. Обновляет матрицу на выстрел
2. Обновляет массивы на выстрел
3. Возвращает обновленные массивы и матрицы
Раненые есть?
Да
Нет
СР > 1
Да
Нет
Лучший ход
Ход = ЛХ
ДБ1
Ход = ДБ1
1. Определение курса при 2 ранении
2. Определении координат выстрела
3. Возвращает ДБ 2 - ход в нужном направлении (таблица 6)
ДБ2
Ход = ДБ2
Убил?
Нет
Да
Выбор стратегии
1. Обновляет счетчик кораблей
2. Обновляет стратегию
3. Возвращает стратегию
Вокруг
1. Заполняет ячейки вокруг уничтоженного корабля
2. Обновляет матрицу на заполнение
3. Обновляет массивы на заполнение (таблица 7)
СР = 0
Обновление выстрела
Лучший ход
1. Определяет ЛХ в зависимости от СТР
2. Выдает следующий ЛХ (таблица 1, 2, 3, 4)
Ход = ЛХ
Обновить СР
Обновление выстрела
СР > 1
Да
Нет
ДБ2
Ход = ДБ2
1. Определение направления при 1 ранении
2. Возвращает ДБ 1 - ход в нужном направлении (таблица 5)
ДБ1
Ход = ДБ1
Отправить ход на сервер
Конец

Таблица 7 - обновление поля вокруг

	0	1	2	3	4	5	6	7	8	9
0									1	1
1									1	x
2	1	1	1	1	1				1	x
3	x	x	x	x	1				1	x
4	1	1	1	1	1				1	1
5										
6										
7					1	1	1	1	1	
8					1	x	x	x	1	
9					1	1	1	1	1	

🟩	Попадание
🟥	Выстрел
🟦	Отметка вокруг корабля

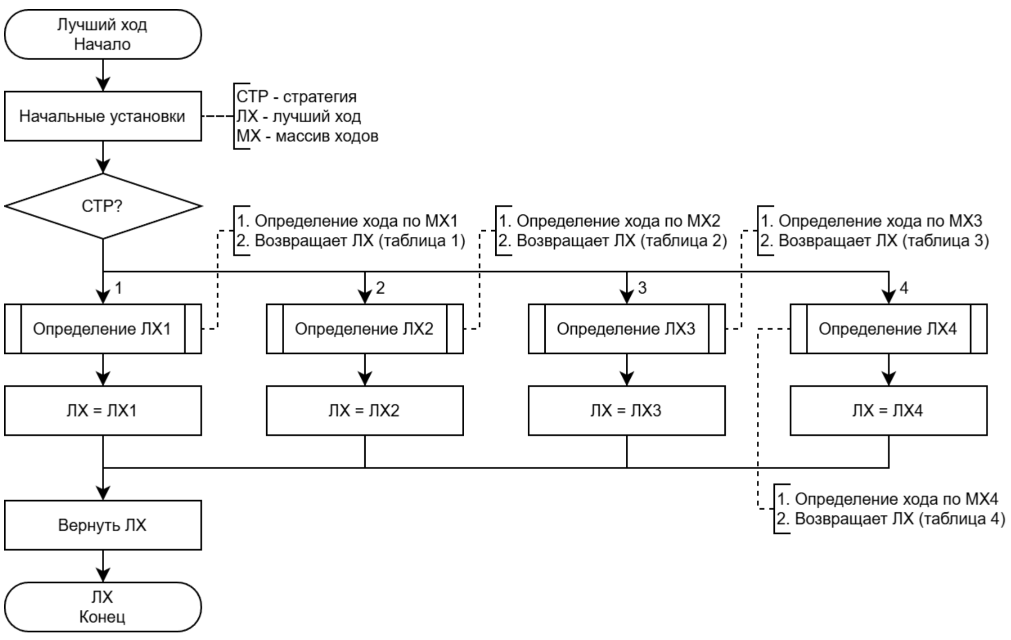

Лучший ход
Начало
Начальные установки
СТР - стратегия
ЛХ - лучший ход
МХ - массив ходов
СТР?
1. Определение хода по МХ1
2. Возвращает ЛХ (таблица 1)
1. Определение хода по МХ2
2. Возвращает ЛХ (таблица 2)
1. Определение хода по МХ3
2. Возвращает ЛХ (таблица 3)
1
2
3
4
Определение ЛХ1
Определение ЛХ2
Определение ЛХ3
Определение ЛХ4
ЛХ = ЛХ1
ЛХ = ЛХ2
ЛХ = ЛХ3
ЛХ = ЛХ4
Вернуть ЛХ
1. Определение хода по МХ4
2. Возвращает ЛХ (таблица 4)
ЛХ
Конец

Таблица 1 - массив ходов МХ1

	0	1	2	3	4	5	6	7	8	9
0				1				2		
1			3				4			
2		5				6				7
3	8				9				10	
4				11				12		
5			13				14			
6		15				16				17
7	18				19				20	
8				21				22		
9			23				24			

Таблица 2 - массив ходов МХ2

	0	1	2	3	4	5	6	7	8	9
0			1			2			3	
1		4			5			6		
2	7			8			9			10
3			11			12			13	
4		14			15			16		
5	17			18			19			20
6			21			22			23	
7		24			25			26		
8	27			28			29			30
9			31			32			33	

Таблица 3 - массив ходов МХ3

	0	1	2	3	4	5	6	7	8	9
0	1		2		3		4		5	
1		6		7		8		9		10
2	11		12		13		14		15	
3		16		17		18		19		20
4	21		22		23		24		25	
5		26		27		28		29		30
6	31		32		33		34		35	
7		36		37		38		39		40
8	41		42		43		44		45	
9		46		47		48		49		50

Таблица 4 - массив ходов МХ4

	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	11	12	13	14	15	16	17	18	19	20
2	21	22	23	24	25	26	27	28	29	30
3	31	32	33	34	35	36	37	38	39	40
4	41	42	43	44	45	46	47	48	49	50
5	51	52	53	54	55	56	57	58	59	60
6	61	62	63	64	65	66	67	68	69	70
7	71	72	73	74	75	76	77	78	79	80
8	81	82	83	84	85	86	87	88	89	90
9	91	92	93	94	95	96	97	98	99	100

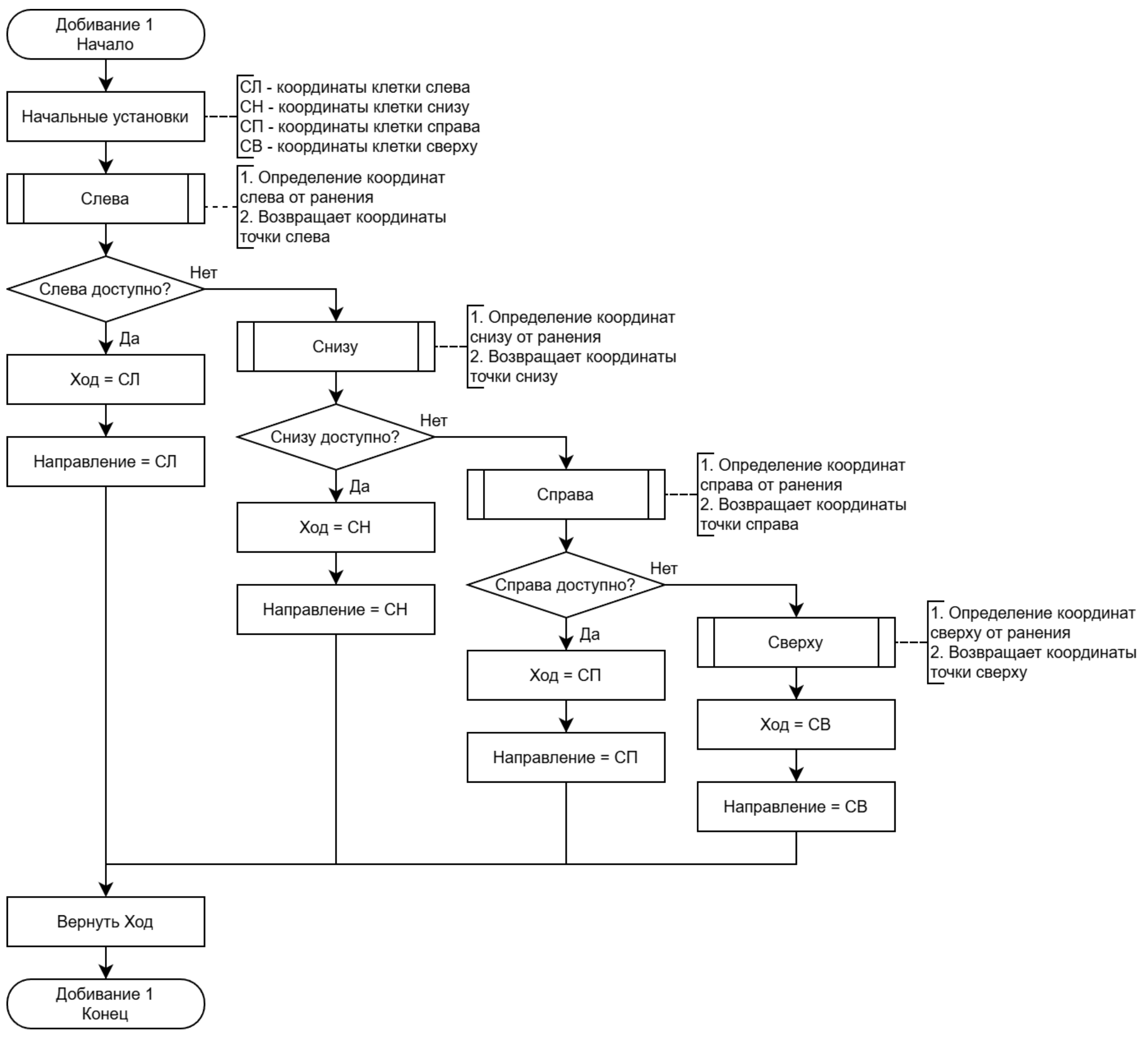

Добивание 1
Начало
Начальные установки
СЛ - координаты клетки слева
СН - координаты клетки снизу
СП - координаты клетки справа
СВ - координаты клетки сверху
Слева
1. Определение координат слева от ранения
2. Возвращает координаты точки слева
Слева доступно?
Нет
Да
Ход = СЛ
Направление = СЛ
Снизу
1. Определение координат снизу от ранения
2. Возвращает координаты точки снизу
Снизу доступно?
Нет
Да
Ход = СН
Направление = СН
Справа
1. Определение координат справа от ранения
2. Возвращает координаты точки справа
Справа доступно?
Нет
Да
Ход = СП
Направление = СП
Сверху
1. Определение координат сверху от ранения
2. Возвращает координаты точки сверху
Ход = СВ
Направление = СВ
Вернуть Ход
Добивание 1
Конец

Таблица 5 - добивание 1, выбор направления выстрела

	0	1	2	3	4	5	6	7	8	9
0	x	2			1	x	3			3
1	1					2			1	x
2			4							2
3		1	x	3						
4			2							
5	3									
6	x	2								
7	1									
8										2
9									1	x

🟩 Попадание

🟥 Выстрел

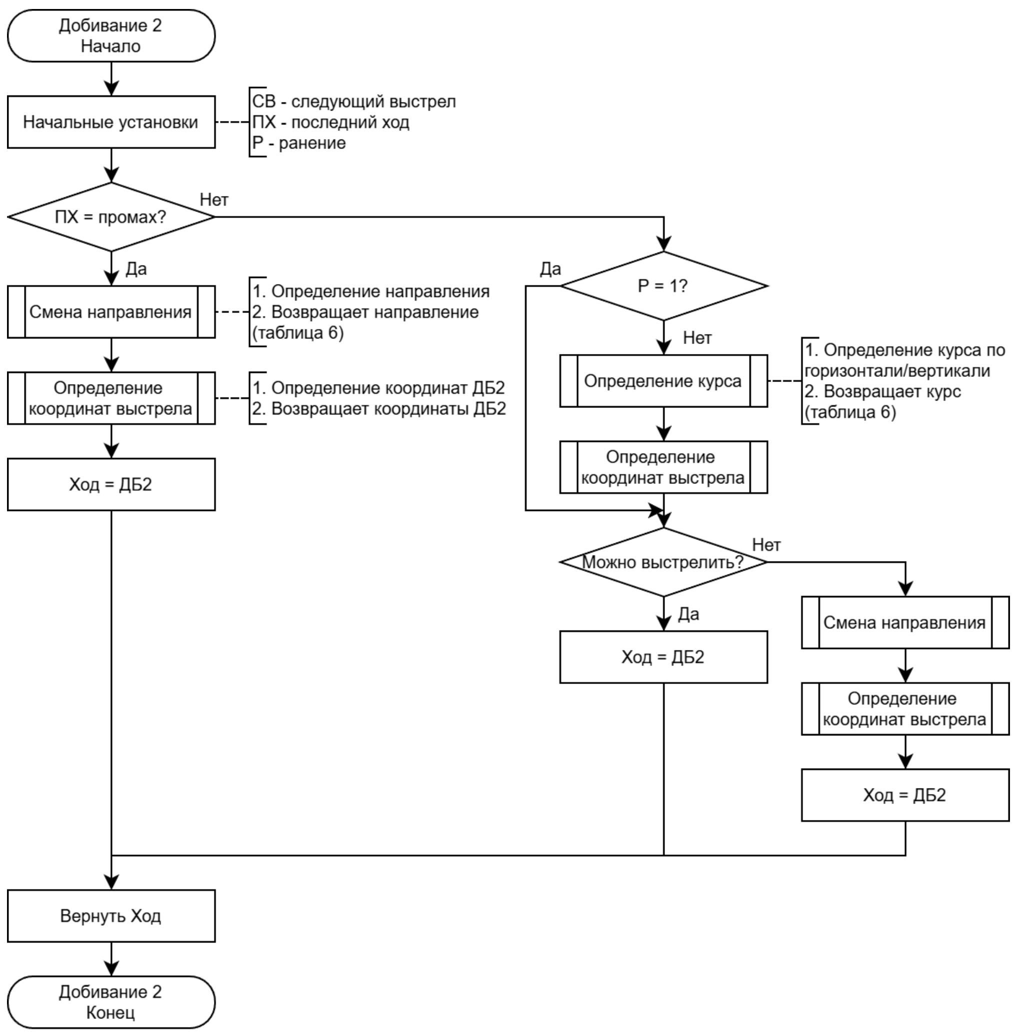
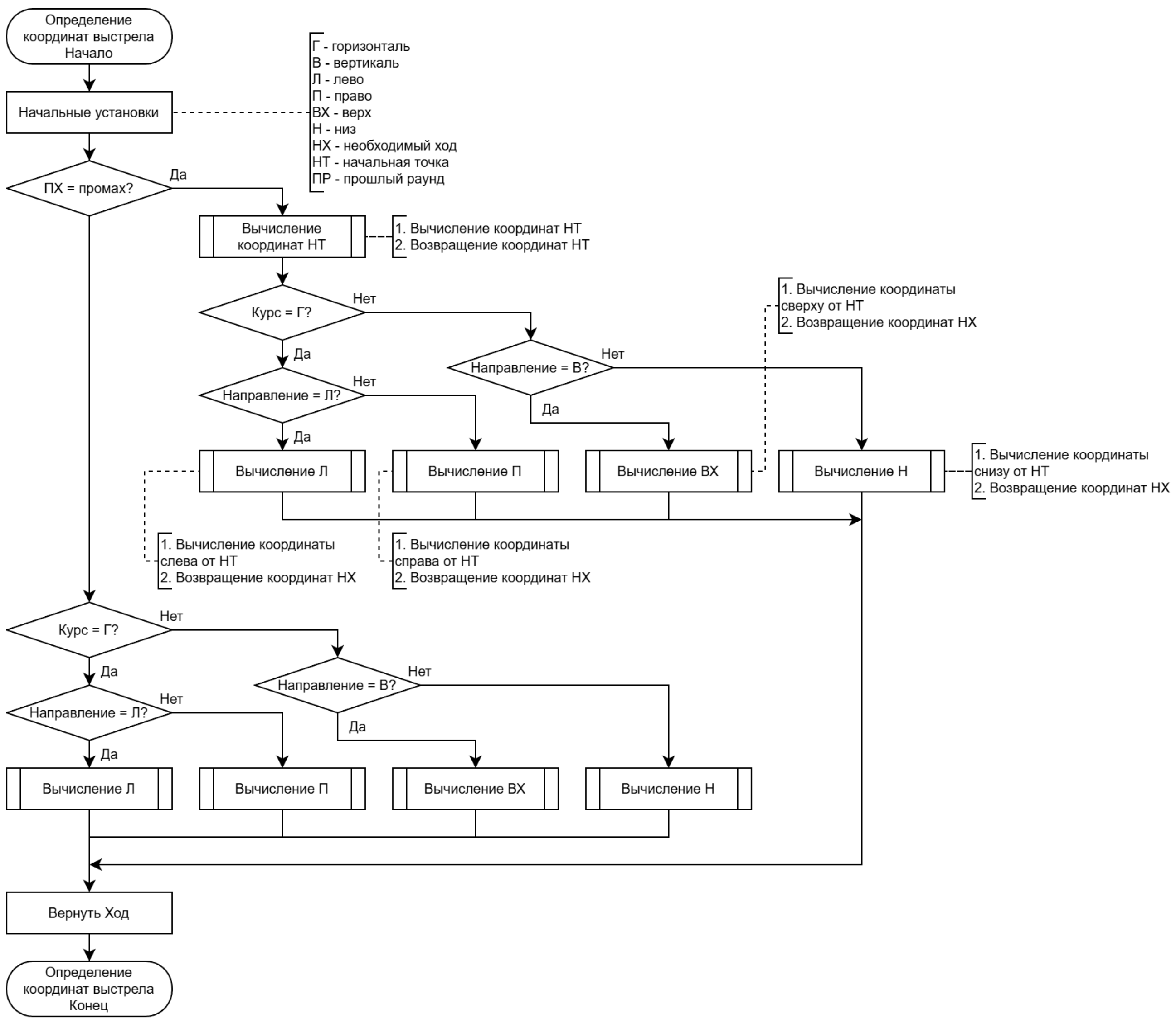


Таблица 6 - добивание 2, выбор направления выстрела

	0	1	2	3	4	5	6	7	8	9
0								2		
1								1		
2			нач					x		
3	1	x	x	2				x	нач	
4								3		
5	нач									
6	x	x	1	2						
7										
8										
9										

🟩	Попадание
🟥	Выстрел
🟦	Промах



Определение координат выстрела Начало
Начальные установки
Г - горизонталь
В - вертикаль
Л - лево
П - право
ВХ - верх
Н - низ
НХ - необходимый ход
НТ - начальная точка
ПР - прошлый раунд
ПХ = промах?
Да
Вычисление координат НТ
1. Вычисление координат НТ
2. Возвращение координат НТ
Курс = Г?
Нет
Да
Направление = В?
Нет
Да
Направление = Л?
Нет
Да
1. Вычисление координаты сверху от НТ
2. Возвращение координат НХ
Вычисление Л
Вычисление П
Вычисление ВХ
Вычисление Н
1. Вычисление координаты снизу от НТ
2. Возвращение координат НХ
1. Вычисление координаты слева от НТ
2. Возвращение координат НХ
1. Вычисление координаты справа от НТ
2. Возвращение координат НХ
Курс = Г?
Нет
Да
Направление = В?
Нет
Да
Направление = Л?
Нет
Да
Вычисление Л
Вычисление П
Вычисление ВХ
Вычисление Н
Вернуть Ход
Определение координат выстрела Конец

Исходный код программы, написанный на языке Python представлен ниже:

```
SHOT_RESULT_EMPTY: int = 0
SHOT_RESULT_DAMAGE: int = 2
SHOT_RESULT_KILL: int = 3

_my_map: list[list[int]] = []
_opponent_map: list[list[int]] = []
_shot_history: list[list[int]] = []

_ship_count: dict[int, int] = {1: 4, 2: 3, 3: 2, 4: 1}

_moves_for_4: list[tuple[int, int]] = []
_moves_for_3: list[tuple[int, int]] = []
_moves_for_2: list[tuple[int, int]] = []
_moves_for_1: list[tuple[int, int]] = []

_hunt_mode: bool = False
_last_hit: tuple[int, int] = (-1, -1)
_hit_list: list[tuple[int, int]] = []


def _initialize_move_arrays() -> None:
    global _moves_for_4, _moves_for_3, _moves_for_2, _moves_for_1

    _moves_for_4 = []
    for i in range(10):
        for j in range(10):
            if j + 3 < 10:
                _moves_for_4.append((i, j))
            if i + 3 < 10:
                _moves_for_4.append((i, j))
    _moves_for_4 = sorted(set(_moves_for_4))

    _moves_for_3 = []
    for i in range(10):
        for j in range(10):
            if j + 2 < 10:
                _moves_for_3.append((i, j))
            if i + 2 < 10:
                _moves_for_3.append((i, j))
    _moves_for_3 = sorted(set(_moves_for_3))

    _moves_for_2 = []
    for i in range(10):
        for j in range(10):
            if j + 1 < 10:
                _moves_for_2.append((i, j))
            if i + 1 < 10:
                _moves_for_2.append((i, j))
    _moves_for_2 = sorted(set(_moves_for_2))

    _moves_for_1 = [(i, j) for i in range(10) for j in range(10)]
    _moves_for_1.sort()


def _is_valid_coordinate(x: int, y: int) -> bool:
    return 0 <= x < 10 and 0 <= y < 10


def _can_shoot_at(x: int, y: int) -> bool:
    return _is_valid_coordinate(x, y) and _shot_history[x][y] == 0


def _remove_move_from_all_arrays(x: int, y: int) -> None:
    global _moves_for_4, _moves_for_3, _moves_for_2, _moves_for_1
    _moves_for_4 = [(i, j) for (i, j) in _moves_for_4 if not (i == x and j == y)]
    _moves_for_3 = [(i, j) for (i, j) in _moves_for_3 if not (i == x and j == y)]
    _moves_for_2 = [(i, j) for (i, j) in _moves_for_2 if not (i == x and j == y)]
    _moves_for_1 = [(i, j) for (i, j) in _moves_for_1 if not (i == x and j == y)]


def _get_best_strategic_move() -> tuple[int, int]:
    if _ship_count[4] > 0:
        for move in _moves_for_4:
            if _can_shoot_at(*move):
                return move
    if _ship_count[3] > 0:
        for move in _moves_for_3:
            if _can_shoot_at(*move):
                return move
    if _ship_count[2] > 0:
        for move in _moves_for_2:
            if _can_shoot_at(*move):
                return move
    if _ship_count[1] > 0:
        for move in _moves_for_1:
            if _can_shoot_at(*move):
                return move
    for i in range(10):
        for j in range(10):
            if _can_shoot_at(i, j):
                return i, j
    return 0, 0


def _hunt_finish_1() -> tuple[int, int]:
    last_hit = _hit_list[0]
    directions = [
        (last_hit[0] - 1, last_hit[1]),
        (last_hit[0], last_hit[1] + 1),
        (last_hit[0] + 1, last_hit[1]),
        (last_hit[0], last_hit[1] - 1)
    ]
    for d in directions:
        if _can_shoot_at(*d):
            return d
    global _hunt_mode
    _hunt_mode = False
    return _get_best_strategic_move()


def _hunt_finish_2() -> tuple[int, int]:
    first_hit = _hit_list[0]
    last_hit = _hit_list[-1]

    if first_hit[0] == last_hit[0]:
        dy = 1 if last_hit[1] > first_hit[1] else -1
        dx = 0
```

```
            next_cell = (last_hit[0], last_hit[1] + dy)
            if _can_shoot_at(*next_cell):
                return next_cell
            else:
                prev_cell = (first_hit[0], first_hit[1] - dy)
                if _can_shoot_at(*prev_cell):
                    return prev_cell
        else:
            dx = 1 if last_hit[0] > first_hit[0] else -1
            dy = 0
            next_cell = (last_hit[0] + dx, last_hit[1])
            if _can_shoot_at(*next_cell):
                return next_cell
            else:
                prev_cell = (first_hit[0] - dx, first_hit[1])
                if _can_shoot_at(*prev_cell):
                    return prev_cell

        global _hunt_mode
        _hunt_mode = False
        return _get_best_strategic_move()


    def _mark_area_around_ship() -> None:
        if not _hit_list:
            return

        xs = [hit[0] for hit in _hit_list]
        ys = [hit[1] for hit in _hit_list]
        min_x, max_x = min(xs), max(xs)
        min_y, max_y = min(ys), max(ys)

        for x in range(min_x - 1, max_x + 2):
            for y in range(min_y - 1, max_y + 2):
                if _is_valid_coordinate(x, y) and _shot_history[x][y] == 0:
                    _shot_history[x][y] = 1
                    _remove_move_from_all_arrays(x, y)


    def _initialize_game_state() -> None:
```

```
    global _my_map, _opponent_map, _shot_history
    _my_map = [[0 for _ in range(10)] for _ in range(10)]
    _opponent_map = [[0 for _ in range(10)] for _ in range(10)]
    _shot_history = [[0 for _ in range(10)] for _ in range(10)]

    global _ship_count
    _ship_count = {1: 4, 2: 3, 3: 2, 4: 1}

    _initialize_move_arrays()

    global _hunt_mode, _hit_list
    _hunt_mode = False
    _hit_list = []


def _generate_fixed_map() -> None:
    global _my_map
    _my_map = [[0 for _ in range(10)] for _ in range(10)]

    _my_map[6][1] = 1
    _my_map[7][1] = 1
    _my_map[8][1] = 1
    _my_map[9][1] = 1

    _my_map[1][7] = 1
    _my_map[1][8] = 1
    _my_map[1][9] = 1

    _my_map[3][4] = 1
    _my_map[4][4] = 1
    _my_map[5][4] = 1

    _my_map[0][2] = 1
    _my_map[1][2] = 1

    _my_map[3][0] = 1
    _my_map[4][0] = 1

    _my_map[9][6] = 1
    _my_map[9][7] = 1
    _my_map[0][5] = 1
    _my_map[2][8] = 1
    _my_map[5][7] = 1
    _my_map[7][9] = 1


def _update_strategy(result_code: int) -> None:
    global _hunt_mode, _hit_list, _ship_count

    if result_code == SHOT_RESULT_DAMAGE:
        _hunt_mode = True
        _hit_list.append(_last_hit)
    elif result_code == SHOT_RESULT_KILL:
        _mark_area_around_ship()
        if _hit_list:
            xs = [h[0] for h in _hit_list]
            ys = [h[1] for h in _hit_list]
            ship_size = max(max(xs) - min(xs), max(ys) - min(ys)) + 1
            if 1 <= ship_size <= 4:
                _ship_count[ship_size] -= 1
        _hunt_mode = False
        _hit_list = []


def set_parameters(set_count: int) -> None:
    pass


def on_game_start() -> None:
    _initialize_game_state()


def on_set_start() -> None:
    global _shot_history, _hunt_mode, _hit_list, _ship_count
    _shot_history = [[0 for _ in range(10)] for _ in range(10)]
    _generate_fixed_map()
    _initialize_move_arrays()
    _hunt_mode = False
    _hit_list = []
    _ship_count = {1: 4, 2: 3, 3: 2, 4: 1}


def get_map() -> list[list[int]]:
    return _my_map


def shoot() -> tuple[int, int]:
    global _last_hit, _hunt_mode

    if _hunt_mode:
        if len(_hit_list) == 1:
            coords = _hunt_finish_1()
        else:
            coords = _hunt_finish_2()
    else:
        coords = _get_best_strategic_move()

    _last_hit = coords
    x, y = coords
    if _is_valid_coordinate(x, y):
        _shot_history[x][y] = 1
        _remove_move_from_all_arrays(x, y)

    return coords


def shot_result(result_code: int) -> None:
    _update_strategy(result_code)


def on_opponent_shot(cell: tuple[int, int]) -> None:
    x, y = cell
    if _is_valid_coordinate(x, y):
        _opponent_map[x][y] = 1


def on_set_end() -> None:
    pass
```

```
def on_game_end() -> None:
    pass
```

Вывод

В ходе выполнения лабораторной работы разработан алгоритм работы бота для игры «Морской бой», а также разработана программа на языке Python, реализующий данный алгоритм.