

Санкт-Петербургский Политехнический Университет Петра
Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Программирование

Отчет по курсовой работе
Модель "Хищник - жертва"

Работу
выполнил:
Жуйков А.А.
Группа:
13501/4
Преподаватель:
Вылегжанина
К.Д.

Санкт-Петербург
2016

Содержание

1	Модель Хищник-жертва	2
1.1	Задание	2
1.2	Концепция	2
1.3	Диаграмма прецедентов использования	2
2	Проектирование приложения, реализующего модель Хищник-жертва	3
2.1	Библиотека	3
3	Реализация модели Хищник - жертва	4
3.1	Среда разработки	4
3.2	Консольное приложение	4
3.3	Библиотека	5
3.4	Графическое приложение	6
4	Процесс обеспечения качества и тестирование	8
4.1	Просмотр кода	8
4.2	Демонстрации	9
4.3	Непрерывная интеграция	9
4.4	Тестирование	9
5	Вывод	10
6	Приложение 1. Листинги кода	10
6.1	Библиотека	10
6.2	Консольное приложение	44
6.3	Графическое приложение	55
6.4	Модульные тесты	74

1 Модель Хищник-жертва

1.1 Задание

На прямоугольном поле случайным образом размещаются "хищники" и "жертвы" после чего они поочередно делают ходы. Ход жертвы - случайное перемещение на соседнюю клетку, раз в несколько ходов жертва порождает еще одну жертву на соседней клетке. Ход хищника - уничтожение жертвы на соседней клетке, если это возможно, иначе - случайное перемещение на соседнюю клетку. Уничтожив несколько жертв, хищник порождает еще одного хищника на соседней клетке. Оставшись без еды на несколько ходов, хищник умирает. Реализовать на экране процесс борьбы хищников и жертв.

1.2 Концепция

На прямоугольном поле случайным образом размещаются "хищники" и "жертвы" после чего они поочередно делают ходы. Ход жертвы - случайное перемещение на соседнюю клетку. Раз в несколько ходов жертва порождает еще одну жертву на соседней клетке, если у нее достаточно энергии. Чтобы выжить и получить энергию, жертве необходимо питаться. На случайной клетке поля "вырастают" растения - пища для жертвы. Ход хищника - уничтожение жертвы на соседней клетке, если это возможно, иначе - случайное перемещение на соседнюю клетку. Уничтожив несколько жертв, хищник порождает еще одного хищника на соседней клетке. Оставшись без еды на несколько ходов, хищник умирает. Реализовать на экране процесс борьбы хищников и жертв.

1.3 Диаграмма прецедентов использования

На рис 1 показана диаграмма прецедентов использования.

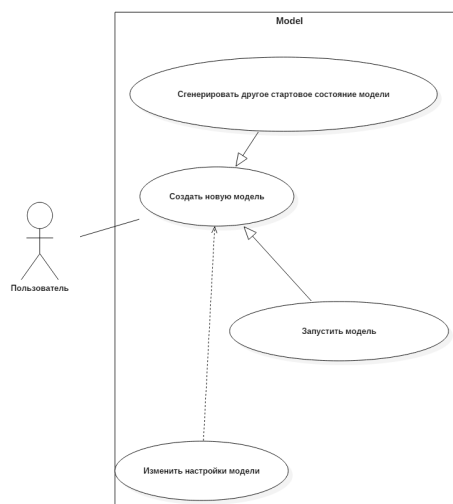


Рис. 1: Диаграмма прецедентов использования

2 Проектирование приложения, реализующего модель Хищник-жертва

Приложение позволяет задавать следующие настройки модели: начальные количества хищников и жертв, длина и высота поля, количество ходов хищника без еды.

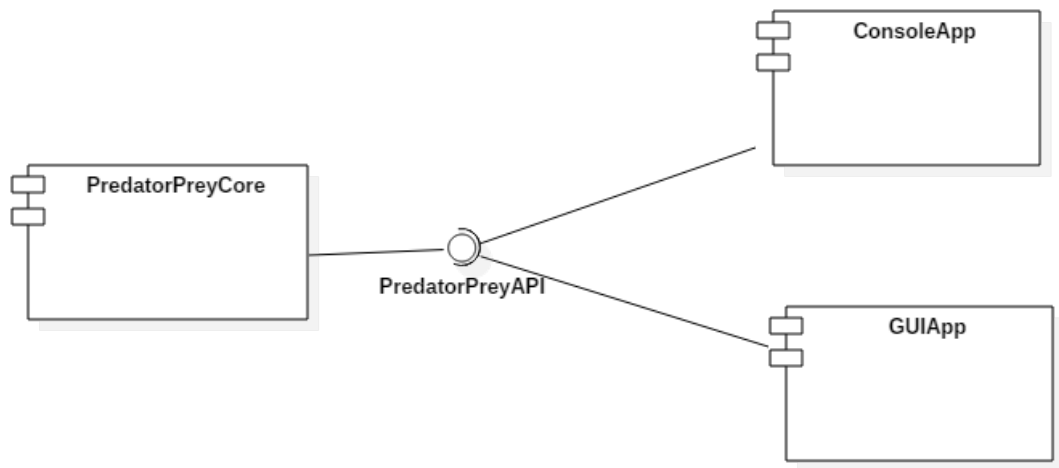


Рис. 2: Диаграмма прецедентов использования

2.1 Библиотека

Библиотека - ядро приложения. Здесь содержатся основные классы, необходимые для представления модели.

В API выделены следующие методы:

- Метод, создающий хищников на поле, количество которых задано в настройках.
- Метод, создающий жертв на поле, количество которых задано в настройках.
- Методы, удаляющие умерших хищников и жертв в конце хода.
- Методы, передвигающие всех хищников и жертв на поле.
- Метод, возвращающий true, если хищники или жертвы исчезли с поля и false в обратном случае.
- Методы, возвращающие текущее количество хищников и жертв на поле.
- Методы, возвращающие текущие день и время модели, необходимые для подсчета времени существования модели. 8. Метод, возвращающий указатель на поле модели.

3 Реализация модели Хищник - жертва

3.1 Среда разработки

Операционная система Windows 8.1

Интегрированная среда разработки Qt Creator 3.6.1

Стандарт C++, C++11

Компилятор MinGW 4.9.2

Система документирования Doxygen 1.8.8

Утилита для статического анализа кода Cppcheck 1.67

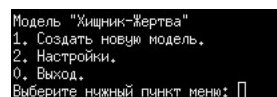
3.2 Консольное приложение

Консольное приложение предоставляет пользователю всю функциональность ядра и позволяет работать с моделью в консоли.

Основные классы, выделенные в консольном приложении:

- Класс ConsoleApp. Создает модель, настройки модели, организует консольное взаимодействие с пользователем. Здесь же задается промежуток времени для вывода состояния модели.
- Класс ConsoleDialog. Содержит консольные меню для взаимодействия с пользователем: меню настроек, главное меню. Реализована обработка неверного ввода.
- Класс ConsoleDrawer. Выводит в консоль состояние модели, а также другую информацию: количество хищников и жертв на поле, время и день модели, легенду.

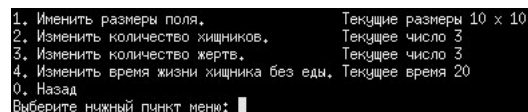
На рис 3 представлено главное меню приложения. Есть возможность начать новую модель, изменить ее настройки и выйти из приложения.



```
Модель "Хищник-Жертва"
1. Создать новую модель.
2. Настройки.
0. Выход.
Выберите нужный пункт меню: █
```

Рис. 3: Главное меню консольного приложения

На рис 4 показано меню настроек с возможностью выбора, что изменить. Справа от настройки отображается ее текущее состояние.



```
1. Изменить размеры поля.           Текущие размеры 10 x 10
2. Изменить количество хищников.     Текущее число 3
3. Изменить количество жертв.        Текущее число 3
4. Изменить время жизни хищника без еды. Текущее время 20
0. Назад
Выберите нужный пункт меню: █
```

Рис. 4: Настройки модели в консольном приложении

На рис 5 – поле модели с находящимися на нем хищниками и жертвами. Над полем выводится легенда и текущее количество агентов.



Рис. 5: Поле модели в консольном приложении

3.3 Библиотека

Основные классы, выделенные в библиотеке:

- Класс Model. Реализует методы, заявленные в API. Содержит поле модели, класс с векторами хищников и жертв, текущие время и день, а также указатель на используемые настройки.
- Класс Settings. Содержит настройки модели: высота и длина поля, количество хищников и жертв на поле, время жизни хищника без еды и другие.
- Класс Field. Класс представляет поле модели. Реализуется в виде вектора векторов позиций поля (enum class Position). Присутствуют методы, возвращающие высоту и длину поля, статус указанной клетки, свободное направление для указанной клетки. В классе определены целочисленные константы, в которых записаны максимальный и минимальный размеры поля.
- Класс Units. Содержит вектора указателей на хищников и жертв.
- Класс Animal – базовый класс для классов хищников и жертв. Содержит поля, необходимые для представления агентов: время нахождения на поле, энергия, направление для следующего хода, указатель на поле и другие. А также общие методы: метод выбора направления следующего хода, метод перемещения агента по текущему направлению, метод выбора случайного направления и другие.
- Класс Predator представляет хищника в модели. Класс реализует полиморфные методы класса Animal и добавляет новые методы и поля к базовому классу: метод уничтожающий жертву, метод поиска жертвы, метод создающий нового хищника, метод передвигающий хищника. Также содержит поля: указатель на жертву – текущая цель, указатель на Units.
- Класс Prey представляет жертву в модели. Как и в классе Predator, здесь присутствует реализация полиморфных методов из класса Animal.

3.4 Графическое приложение

На рис 6 представлено главное окно приложения. Пользователю, как и в консольном приложении, предоставляется возможность начать новую модель, изменить ее настройки и выйти.



Рис. 6: Главное меню графического приложения

На рис 7 – окно настроек приложения. Все начальные параметры модели можно изменить, после чего сохранить изменения. Если выйти из настроек, не нажав на кнопку «сохранить», то настройки не изменятся.

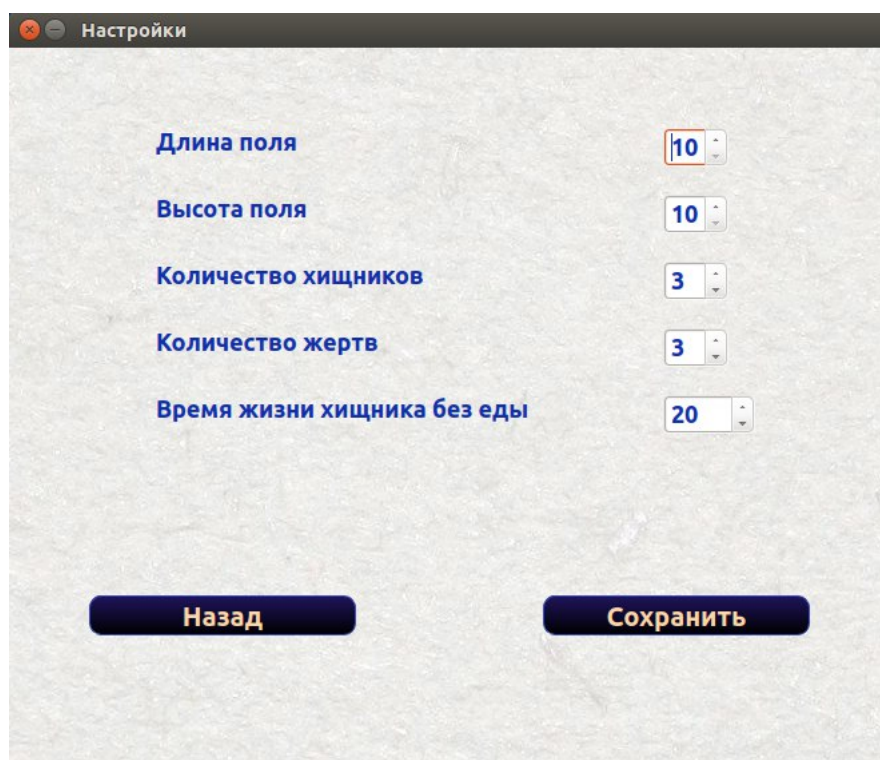


Рис. 7: Настройки модели в графическом приложении

Состояние поля модели, и его текущее состояние представлены на рис 8



Рис. 8: Представление модели в графическом приложении

Основные классы, выделенные в графическом приложении.

- Класс MainMenu. Главное окно приложения. Присутствуют кнопки «Новая модель», «Настройки», «Выход». При попытке выхода появляется окно подтверждения.
- Класс SettingsWindow. Окно для изменения настроек модели. Можно выйти как без сохранения настроек, так и с сохранением.
- Класс ModelWindow. Окно для представления модели. Содержит фрейм поля, в котором выводится состояние поля на каждом шаге моделирования, и фрейм статуса модели, где содержится информация о текущем состоянии модели. Также присутствуют кнопки «Старт», «Сгенерировать», «Выйти в меню».

4 Процесс обеспечения качества и тестирование

4.1 Просмотр кода

В ходе проектирования дважды была проведена проверка исходного кода программы с целью обнаружения и исправления ошибок (code review). В результате было получено около 80-ти замечаний в каждом.

Большая часть замечаний исправлена.

4.2 Демонстрации

Всего было проведено три демонстрации:

- Демонстрация №1. Отрисовка модели в консоль, изменение настроек.

Замечания:

- Сделать один язык во всем приложении.
- Исправить время модели, создающейся с новыми параметрами.

- Демонстрация №2. Движение, порождение/исчезновение хищников.

Замечания:

- Непонятные фразы результата моделирования: "Хищники победили" "Жертвы победили".

- Демонстрация №3. Последняя демонстрация перед релизом консольного приложения

Замечания:

- Не выходить из меню настроек после их изменения.
- Добавить обработку ввода.
- Добавить легенду
- Выводить текущую статистику

Все недочеты исправлены, учтены пожелания других участников демонстраций.

4.3 Непрерывная интеграция

Непрерывная интеграция была осуществлена при помощи приложения jenkins, установленного на сервере с операционной системой Ubuntu. На сервере код приложения проверялся следующими и другими утилитами, строились графики:

Sppcheck – утилита для статического анализа кода. Благодаря ей были найдены и исправлены многие стилистические ошибки.

Valgrind – утилита для обнаружения утечек памяти. С ее помощью быстро исправлялись ошибки в работе с памятью.

Gcov – утилита, которая считает процент покрытия кода тестами. Во время работы процент повысился до 70

4.4 Тестирование

Приложение содержит модульные тесты. Протестированы некоторые основные классы. Имеется большое количество тестов класса Predator: проверяется правильность нахождения направления к жертве, порождение, движение, исчезновение хищника. В ходе работы над проектом покрытие тестами не опускалось ниже 40

5 Вывод

Курсовая работа этого семестра – это отличная возможность ближе познакомиться с языком, глубже его понять. Проведенные семинары по эффективному использованию STL, новым стандартам языка программирования подкрепляют полученные знания и открывают новые возможности использования средств языка.

Работа над приложением будет продолжаться. Планируется добавить еду для жертв; новые тесты, привести в порядок имеющиеся. Хищники должны ускоряться во время преследования жертвы, при этом тратить энергию. Возможно, в процессе улучшения появятся новые идеи для оптимизации алгоритмов ядра.

Таким образом, во время работы над проектом, автор приобрел большой практический опыт в области объектно-ориентированного проектирования на языке C++.

6 Приложение 1. Листинги кода

6.1 Библиотека

```
1 #ifndef MODELAPI_H
2 #define MODELAPI_H
3 #include "units.h"
4 #include "field.h"
5 #include <vector>
6
7 /**
8  * @brief класс, предоставляющий методы ядра
9  */
10 class ModelAPI
11 {
12 public:
13
14     /**
15      * @brief метод, возвращающий указатель на поле модели
16      */
17     virtual Field* getField() noexcept = 0;
18
19     /**
20      * @brief метод, возвращающий текущее время
21      */
22     virtual int getTime() const noexcept = 0;
23
24     /**
25      * @brief метод, возвращающий текущий день
26      */
27     virtual int getDay() const noexcept = 0;
28
29     /**
30      * @brief метод, возвращающий количество хищников на поле
31      */
32     virtual unsigned int getPredatorsNum() const noexcept = 0;
33
34     /**
35      * @brief метод, возвращающий количество жертв на поле
36      */
37     virtual unsigned int getPreysNum() const noexcept = 0;
```

```

38
39 /**
40  * @brief метод, проверяющий, не исчезли ли хищники или жертвы
41  */
42 virtual bool isEnd() const noexcept = 0;
43
44 /**
45  * @brief метод, создающий хищников
46  */
47 virtual void createPredators() noexcept = 0;
48
49 /**
50  * @brief метод, создающий жертв
51  */
52 virtual void createPreys() noexcept = 0;
53
54 /**
55  * @brief метод, удаляющий умерших хищников после хода
56  */
57 virtual void removePredators() noexcept = 0;
58
59 /**
60  * @brief метод, удаляющий умерших жертв после хода
61  */
62 virtual void removePreys() noexcept = 0;
63
64 /**
65  * @brief метод, передвигающий жертв
66  */
67 virtual void movePreys() noexcept = 0;
68
69 /**
70  * @brief метод, передвигающий хищников
71  */
72 virtual void movePredators() noexcept = 0;
73
74 void saveModel();
75 void loadModel();
76
77 virtual ~ModelAPI() {}
78 };
79 #endif // MODELAPI_H

```

```

1 #ifndef MODEL_H
2 #define MODEL_H
3 #include "modelapi.h"
4 #include <vector>
5
6 //TODO указывать слово override для перекрывающих функций
7 class Model : public ModelAPI
8 {
9     Settings *settings;
10    int model_time;
11    int model_day;
12    bool has_changed;
13    Field field;
14    /**
15     * @brief Units — класс, в котором содержатся векторы юнитов:
16     * ↪ хищников и жертв
17     */
18    Units units;
19
20    void incModelTime() noexcept;
21
22 public:
23    explicit Model(Settings *settings) noexcept;
24    Field* getField() noexcept { return &field; }
25    int getTime() const noexcept { return model_time; }
26    int getDay() const noexcept { return model_day; }
27    unsigned int getPredatorsNum() const noexcept { return this->
28    ↪ units.predators.size(); }
29    unsigned int getPreysNum() const noexcept { return units.preys.
30    ↪ size(); }
31    void movePreys() noexcept;
32    void movePredators() noexcept;
33    bool isEnd() const noexcept;
34    void createPredators() noexcept;
35    void createPreys() noexcept;
36    void removePredators() noexcept;
37    void removePreys() noexcept;
38    void remove() noexcept;
39
40    void saveModel();
41    void loadModel();
42 };
43 #endif // MODEL_H

```

```

1 #include "model.h"
2 #include <algorithm>
3 #include <ctime>
4 #include <cstdlib>
5
6 Model::Model(Settings *settings) noexcept:
7     settings(settings),
8     model_time(0),
9     model_day(0),
10    has_changed(false),
11    field(settings->getFieldHeight(), settings->getFieldLength())
12 {
13     srand(time(0));
14     createPredators();
15     createPreys();

```

```

16 | }
17 |
18 |
19 | bool Model::isEnd() const noexcept
20 | {
21 |     return (units.predators.empty() || units.preys.empty());
22 | }
23 |
24 | void Model::createPredators() noexcept
25 | {
26 |     for(int i = 0; i < settings->getNumOfPredators(); i++) {
27 |         int v = 0;
28 |         int h = 0;
29 |         do {
30 |             v = rand() % settings->getFieldHeight();
31 |             h = rand() % settings->getFieldLength();
32 |         }
33 |         while(field.isEmpty(v, h) == false);
34 |
35 |         new Predator(v, h, &field, &units, settings->
    ↪ getMovesWithoutMeal());
36 |     }
37 | }
38 |
39 | void Model::createPreys() noexcept
40 | {
41 |     for(int i = 0; i < settings->getNumOfPreys(); i++) {
42 |         int v = 0;
43 |         int h = 0;
44 |         do {
45 |             v = rand() % settings->getFieldHeight();
46 |             h = rand() % settings->getFieldLength();
47 |         }
48 |         while (field.isEmpty(v, h) == false);
49 |
50 |         new Prey(v, h, &field, &units);
51 |     }
52 | }
53 |
54 | void Model::movePreys() noexcept
55 | {
56 |     incModelTime();
57 |
58 |     std::vector< Prey* >::iterator last = units.preys.end();
59 |     for (std::vector< Prey* >::iterator i = units.preys.begin(); i
    ↪ != last; ++i) {
60 |         if ((*i)->died == false) (*i)->movePrey();
61 |     }
62 |
63 | }
64 |
65 | void Model::movePredators() noexcept
66 | {
67 |     incModelTime();
68 |
69 |     std::vector< Predator* >::iterator last = units.predators.end()
    ↪ ;
70 |     for (std::vector< Predator* >::iterator i = units.predators.
    ↪ begin(); i != last; ++i) {
71 |         if ((*i)->died == false) (*i)->movePredator();
72 |     }
73 | }

```

```

74
75 void Model::incModelTime() noexcept
76 {
77     if (this->has_changed == false) {
78         this->model_time ++;
79         this->has_changed = true;
80     }
81     else has_changed = false;
82
83     if (this->model_time > 23) {
84         this->model_day ++;
85         this->model_time = 0;
86     }
87 }
88
89 //TODO: не говоря о прочем, этот метод и removePreys очень похожи,
90     ↪ уверена, что используя наследование от animal, можно попытаться
91     ↪ объединить в один полиморфный метод,
92 //это можно обсудить отдельно после основного и очевидного рефакторинга
93 void Model::removePredators() noexcept
94 {
95     for (std::vector< Predator* >::iterator it = units.predators.
96     ↪ begin(); it != units.predators.end(); ++it) {
97         if ( (*it)->died == true ) {
98             delete (*it);
99             (*it) = nullptr;
100         }
101     }
102     units.predators.erase( std::remove(units.predators.begin(),
103     ↪ units.predators.end(), nullptr),
104                           units.predators.end() );
105 }
106
107 void Model::removePreys() noexcept
108 {
109     for (std::vector< Prey* >::iterator it = units.preys.begin();
110     ↪ it != units.preys.end(); ++it) {
111         if ( (*it)->died == true ) {
112             delete (*it);
113             (*it) = nullptr;
114         }
115     }
116     units.preys.erase( std::remove(units.preys.begin(), units.preys
117     ↪ .end(), nullptr),
118                       units.preys.end() );
119 }
120
121 void Model::remove() noexcept
122 {
123     removePredators();
124     removePreys();
125 }

```

```

1 #ifndef ANIMAL_H
2 #define ANIMAL_H
3 #include "field.h"
4
5 /**
6  * @brief класс, от которого наследуются хищники и жертвы
7  */
8
9 class Animal
10 {
11     /**
12      * @brief метод, выбирающий свободное направление;
13      * используется, если переход по выбранному направлению невозможен
14      */
15     void chooseEmptyDirection() noexcept;
16
17 protected:
18
19     /**
20      * @brief DISTANCE_FOR_KILL — дистанция до жертвы, при которой
21      * ↪ можно ее съесть
22      */
23     static constexpr double DISTANCE_FOR_EAT = 1;
24
25     /**
26      * @brief DISTANCE_FOR_TARGET — дистанция для взятия жертвы в цель
27      */
28     static constexpr double DISTANCE_FOR_TARGET = 1.4;
29
30     /**
31      * @brief DELTA — константа, необходимая для сравнения чисел
32      */
33     static constexpr double DELTA = 0.1;
34
35     /**
36      * @brief life_time — счетчик ходов животного на поле
37      */
38     int life_time;
39
40     /**
41      * @brief max_life_time — максимальное время жизни животного без
42      * ↪ еды
43      */
44     int max_life_time;
45
46     /**
47      * @brief energy — текущая энергия животного
48      */
49     int energy;
50
51     /**
52      * @brief has_moved — флаг; используется в случае, когда все четыре
53      * направления заблокированы
54      */
55     bool has_moved;
56
57     /**
58      * @brief direction — текущее направление животного
59      */
60     Direction direction;

```



```

61     * @brief field — указатель на поле, где стоит животное
62     */
63     Field* field;
64
65     /**
66     * @brief метод устанавливает направление, если соответствующая клетка
67     ↪ свободна
68     * @return true, если удалось установить направление
69     */
70     bool setDirection(Direction) noexcept;
71
72     /**
73     * @brief метод, выбирающий случайное направление,
74     * записывает его в direction
75     */
76     void chooseRandomDirection() noexcept;
77
78     /**
79     * @brief метод, выбирающий направление для следующего хода,
80     * записывает его в direction
81     */
82     virtual void directionFinding() noexcept = 0;
83
84     /**
85     * @brief метод, выбирающий направление, в зависимости от положения
86     ↪ цели
87     */
88     virtual void chooseToTargetDirection() noexcept = 0;
89
90     /**
91     * @brief метод, перемещающий животное в направлении direction
92     */
93     virtual void go() noexcept;
94
95 public:
96
97     /**
98     * @brief place — координаты животного на поле
99     */
100    Coordinates place;
101
102    /**
103    * @brief died — флаг; died = true, если животное умерло,
104    * died = false если животное живое
105    */
106    bool died;
107
108    virtual ~Animal() {}
109 };
110 #endif // ANIMAL_H

```

```

1 #include "animal.h"
2 #include <ctime>
3 #include <cstdlib>
4
5 void Animal::chooseEmptyDirection() noexcept
6 {
7     direction = field->whatIsEmpty(place.getV(), place.getH());
8     if (direction == Direction::NO_DIRECTION) {
9         has_moved = true;
10        direction = Direction::UP;

```

```

11     }
12 }
13
14 bool Animal::setDirection(Direction direction) noexcept
15 {
16     switch (direction)
17     {
18         case Direction::UP: {
19             if (field->isEmpty(place.getV() - 1, place.getH())) {
20                 this->direction = Direction::UP;
21                 return true;
22             }
23         }
24         case Direction::DOWN: {
25             if (field->isEmpty(place.getV() + 1, place.getH())) {
26                 this->direction = Direction::DOWN;
27                 return true;
28             }
29         }
30         case Direction::LEFT: {
31             if (field->isEmpty(place.getV(), place.getH() - 1)) {
32                 this->direction = Direction::LEFT;
33                 return true;
34             }
35         }
36         case Direction::RIGHT: {
37             if (field->isEmpty(place.getV(), place.getH() + 1)) {
38                 this->direction = Direction::RIGHT;
39                 return true;
40             }
41         }
42         default: {}
43     }
44     return false;
45 }
46
47 void Animal::chooseRandomDirection() noexcept
48 {
49     int flag = rand() % 4;
50     //TODO: будет понятнее, если в case тоже использовать enum
51     switch (flag) {
52         case 0: {
53             if (setDirection(Direction::UP) == false)
54                 chooseEmptyDirection();
55             break;
56         }
57         case 1: {
58             if (setDirection(Direction::RIGHT) == false)
59                 chooseEmptyDirection();
60             break;
61         }
62         case 2: {
63             if (setDirection(Direction::LEFT) == false)
64                 chooseEmptyDirection();
65             break;
66         }
67         case 3: {
68             if (setDirection(Direction::DOWN) == false)
69                 chooseEmptyDirection();
70             break;
71         }
72     }

```

```
73 | }  
74 |  
75 | void Animal::go() noexcept  
76 | {  
77 |     place.changeToDirection(direction);  
78 | }
```

```

1 #ifndef FIELD_H
2 #define FIELD_H
3 #include "settings.h"
4 #include "coordinates.h"
5 #include <vector>
6
7 /**
8  * @brief Position — перечисление всех возможных состояний клетки поля
9  */
10 enum class Position
11 {
12     EMPTY,
13     PREDATOR,
14     PREY,
15     GRASS
16 };
17
18 /**
19  * @brief класс для представления поля в программе
20  */
21 class Field
22 {
23     std::vector< std::vector<Position> > field;
24
25     /**
26      * @brief height — текущая длина поля
27      */
28     int height;
29
30     /**
31      * @brief length — текущая длина поля
32      */
33     int length;
34
35     /**
36      * @brief проверить, не находится ли позиция за границами поля
37      * @param vertical_position по вертикали
38      * @param horizontal_position по горизонтали
39      * @return
40      */
41     bool checkBoundary(int vertical_position, int
42 ↪ horizontal_position) const;
43 public:
44     /**
45      * @brief MAX_FIELD_SIZE — максимальная длина и высота поля
46      */
47     //TODO: Подозреваю, что это ограничение вызвано только размерами,
48 ↪ которые возможно отрисовать в консоли, предлагаю вынести его отсюда
49     //А вообще ваше поле, кажется, только размером int ограничено, или
50 ↪ максимальной длиной вектора
51     static constexpr int MAX_FIELD_SIZE = 30;
52
53     /**
54      * @brief MIN_FIELD_SIZE — минимальная длина и высота поля
55      */
56     static constexpr int MIN_FIELD_SIZE = 10;
57
58     /**
59      * @brief конструктор с параметрами, создает поле указанных размеров
60      * @param height — высота поля
61      * @param length — длина поля

```

```

60     */
61     Field(int height = 10, int length = 10);
62
63     /**
64     * @brief метод, позволяющий узнать, является ли клетка с данными
65     ↪ координатами пустым
66     * @param v — координата по вертикали
67     * @param h — координата по горизонтали
68     * @return возвращает true, если клетка свободна и false, если
69     ↪ клетка занята
70     */
71     bool isEmpty(int v, int h) const;
72
73     /**
74     * @brief метод, позволяющий установить на клетку с данными
75     ↪ координатами заданный символ
76     * @param v — координата по вертикали
77     * @param h — координата по горизонтали
78     * @param Position — позиция, которую надо установить
79     */
80     void setPosition(int v, int h, Position);
81
82     /**
83     * @brief метод, возвращающий свободное направление хода для
84     ↪ заданной клетки
85     * @param v — координата клетки по вертикали
86     * @param h — координата клетки по горизонтали
87     */
88     Direction whatIsEmpty(int v, int h) const;
89
90     /**
91     * @brief метод, возвращающий значение клетки с заданными
92     ↪ координатами
93     * @param v — координата по вертикали
94     * @param h — координата по горизонтали
95     * @return символ — значение
96     */
97     Position getPosition(int v, int h) const;
98
99     /**
100    * @brief метод, возвращающий длину поля в клетках
101    * @return длина поля
102    */
103    int getLength() const { return this->length; }
104
105    /**
106    * @brief метод, возвращающий высоту поля в клетках
107    * @return высота поля
108    */
109    int getHeight() const { return this->height; }
110
111    /**
112    * @brief перегруженный оператор присваивания; при необходимости,
113    ↪ изменяет размер поля
114    */
115    };
116 #endif // FIELD_H

```

```

1 #include "field.h"
2 #include "badfield.h"
3 #include "badboundary.h"

```

```

4
5 Field::Field(int height, int length)
6 {
7     this->height = height;
8     this->length = length;
9
10    for(int i = 0; i < height; i++){
11        field.push_back(std::vector<Position>(length, Position::
↪ EMPTY));
12    }
13 }
14
15 bool Field::checkBoundary(int v, int h) const
16 {
17     if (v < 0 || v >= height || h < 0 || h >= length){
18         return false;
19     }
20     return true;
21 }
22
23 bool Field::isEmpty(int v, int h) const
24 {
25     if(checkBoundary(v, h) == false){
26         return false;
27     }
28
29     if (this->field[v][h] != Position::EMPTY){
30         return false;
31     }
32
33     return true;
34 }
35
36 void Field::setPosition(int v, int h, Position pos)
37 {
38     if (checkBoundary(v, h) == false){
39         throw BadFieldBoundary(v, h);
40     }
41     this->field[v][h] = pos;
42 }
43
44 Position Field::getPosition(int v, int h) const
45 {
46     if (checkBoundary(v, h) == false){
47         throw BadFieldBoundary(v, h);
48     }
49     return this->field[v][h];
50 }
51
52 Direction Field::whatIsEmpty(int v, int h) const
53 {
54     if (checkBoundary(v, h) == false){
55         throw BadFieldBoundary(v, h);
56     }
57
58     if (isEmpty(v - 1, h) == true) return Direction::UP;
59     if (isEmpty(v, h + 1) == true) return Direction::RIGHT;
60     if (isEmpty(v + 1, h) == true) return Direction::DOWN;
61     if (isEmpty(v, h - 1) == true) return Direction::LEFT;
62     return Direction::NO_DIRECTION;
63 }

```

```

1 #ifndef SETTINGS_H
2 #define SETTINGS_H
3
4 /**
5  * @brief класс, содержащий настройки модели
6  */
7 class Settings
8 {
9     /**
10     * @brief field_length — текущая длина поля
11     */
12     int field_length;
13
14     /**
15     * @brief field_height — текущая ширина высота() поля
16     */
17     int field_height;
18
19     /**
20     * @brief moves_without_meal — текущее время жизни животного без
    ↪ еды
21     */
22     int moves_without_meal;
23
24     /**
25     * @brief min_moves_without_meal — минимальное время жизни
    ↪ животного без еды
26     */
27     int min_moves_without_meal;
28
29     /**
30     * @brief max_moves_without_meal — максимальное время жизни
    ↪ животного без еды
31     */
32     int max_moves_without_meal;
33
34     /**
35     * @brief num_of_predators — текущее число хищников
36     */
37     int num_of_predators;
38
39     /**
40     * @brief num_of_preys — текущее число жертв
41     */
42     int num_of_preys;
43
44     /**
45     * @brief метод, проверяющий число хищников и жертв;
46     * если число больше максимально допустимого при текущих настройках
    ↪ поля,
47     * устанавливается максимально возможное
48     */
49     void checkNumOfUnits();
50
51 public:
52     Settings();
53
54     /**
55     * @brief методы, возвращающие информацию о текущих настройках
56     */
57     int getFieldLength() const { return field_length; }
58

```

```

59     int getFieldHeight() const { return field_height; }
60     int getNumOfPreys() const { return num_of_preys; }
61     int getNumOfPredators() const { return num_of_predators; }
62     int getMovesWithoutMeal() const { return moves_without_meal; }
63     int getMinMovesWithoutMeal() const { return
    ↪ min_moves_without_meal; }
64     int getMaxMovesWithoutMeal() const { return
    ↪ max_moves_without_meal; }
65     int getMaxUnits() const;
66
67     /**
68      * @brief методы, устанавливающие новые настройки;
69      * при необходимости генерируют исключения
70      */
71     void setFieldLength(const int);
72     void setFieldHeight(const int);
73     void setNumOfPredators(const int);
74     void setNumOfPreys(const int);
75     void setMovesWithoutMeal(const int);
76
77 };
78
79 #endif // SETTINGS_H

```

```

1  #include "settings.h"
2  #include "field.h"
3  #include "badfield.h"
4  #include "badnum.h"
5
6  Settings::Settings():
7      field_length(10),
8      field_height(10),
9      moves_without_meal(20),
10     min_moves_without_meal(5),
11     max_moves_without_meal(1000),
12     num_of_predators(3),
13     num_of_preys(3)
14 {}
15
16 int Settings::getMaxUnits() const
17 {
18     int max_num;
19     max_num = std::max(field_height, field_length) * 2;
20
21     return max_num;
22 }
23
24 void Settings::checkNumOfUnits()
25 {
26     if (num_of_predators > getMaxUnits()) {
27         setNumOfPredators(getMaxUnits());
28     }
29     if (num_of_preys > getMaxUnits()) {
30         setNumOfPreys(getMaxUnits());
31     }
32 }
33
34 void Settings::setFieldLength(const int length)
35 {
36     if (length < Field::MIN_FIELD_SIZE || length > Field::
    ↪ MAX_FIELD_SIZE) {
37         throw BadFieldLength(length);

```



```

38     }
39     this->field_length = length;
40     checkNumOfUnits();
41 }
42
43 void Settings::setFieldHeight(const int height)
44 {
45     if (height < Field::MIN_FIELD_SIZE || height > Field::
↪ MAX_FIELD_SIZE) {
46         throw BadFieldHeight(height);
47     }
48     this->field_height = height;
49     checkNumOfUnits();
50 }
51
52 void Settings::setMovesWithoutMeal(const int moves)
53 {
54     if (moves < min_moves_without_meal || moves >
↪ max_moves_without_meal) {
55         throw BadNum(moves, min_moves_without_meal,
↪ max_moves_without_meal);
56     }
57     this->moves_without_meal = moves;
58 }
59
60 void Settings::setNumOfPredators(const int num)
61 {
62     int MAX_NUM = this->getMaxUnits();
63
64     if (num < 1 || num > MAX_NUM) {
65         throw BadNum(num, 1, MAX_NUM);
66     }
67     this->num_of_predators = num;
68 }
69
70 void Settings::setNumOfPreys(const int num)
71 {
72     int MAX_NUM = this->getMaxUnits();
73
74     if (num < 1 || num > MAX_NUM) {
75         throw BadNum(num, 1, MAX_NUM);
76     }
77     this->num_of_preys = num;
78 }

```

```

1 #ifndef UNITS_H
2 #define UNITS_H
3 #include "predator.h"
4 #include "prey.h"
5 #include "grass.h"
6 #include <vector>
7
8 class Predator;
9 class Grass;
10
11 /**
12  * @brief класс для содержания векторов хищников и жертв, а также корма
13  *      ↗ для жертв
14  */
15 class Units
16 {
17 public:
18     /**
19      * @brief preys — вектор указателей на жертву
20      */
21     std::vector< Prey* > preys;
22
23     /**
24      * @brief predators — вектор указателей на хищника
25      */
26     std::vector< Predator* > predators;
27
28     /**
29      * @brief Units
30      */
31     std::vector< Grass* > grass;
32
33     Units();
34     ~Units();
35 };
36 #endif // UNITS_H

```

```

1 #include "units.h"
2 #include <iostream>
3
4 Units::Units()
5 {
6     predators.reserve(1);
7     preys.reserve(1);
8 }
9
10 Units::~~Units()
11 {
12     for (Predator* predator: predators) {
13         delete predator;
14     }
15     for (Prey* prey: preys) {
16         delete prey;
17     }
18 }
19 }

```

```

1 #ifndef ANIMAL_H
2 #define ANIMAL_H
3 #include "field.h"
4
5 /**
6  * @brief класс, от которого наследуются хищники и жертвы
7  */
8
9 class Animal
10 {
11     /**
12      * @brief метод, выбирающий свободное направление;
13      * используется, если переход по выбранному направлению невозможен
14      */
15     void chooseEmptyDirection() noexcept;
16
17 protected:
18
19     /**
20      * @brief DISTANCE_FOR_KILL — дистанция до жертвы, при которой
21      * ↪ можно ее съесть
22      */
23     static constexpr double DISTANCE_FOR_EAT = 1;
24
25     /**
26      * @brief DISTANCE_FOR_TARGET — дистанция для взятия жертвы в цель
27      */
28     static constexpr double DISTANCE_FOR_TARGET = 1.4;
29
30     /**
31      * @brief DELTA — константа, необходимая для сравнения чисел
32      */
33     static constexpr double DELTA = 0.1;
34
35     /**
36      * @brief life_time — счетчик ходов животного на поле
37      */
38     int life_time;
39
40     /**
41      * @brief max_life_time — максимальное время жизни животного без
42      * ↪ еды
43      */
44     int max_life_time;
45
46     /**
47      * @brief energy — текущая энергия животного
48      */
49     int energy;
50
51     /**
52      * @brief has_moved — флаг; используется в случае, когда все четыре
53      * направления заблокированы
54      */
55     bool has_moved;
56
57     /**
58      * @brief direction — текущее направление животного
59      */
60     Direction direction;

```

```

61     * @brief field — указатель на поле, где стоит животное
62     */
63     Field* field;
64
65     /**
66     * @brief метод устанавливает направление, если соответствующая клетка
67     ↪ свободна
68     * @return true, если удалось установить направление
69     */
70     bool setDirection(Direction) noexcept;
71
72     /**
73     * @brief метод, выбирающий случайное направление,
74     * записывает его в direction
75     */
76     void chooseRandomDirection() noexcept;
77
78     /**
79     * @brief метод, выбирающий направление для следующего хода,
80     * записывает его в direction
81     */
82     virtual void directionFinding() noexcept = 0;
83
84     /**
85     * @brief метод, выбирающий направление, в зависимости от положения
86     ↪ цели
87     */
88     virtual void chooseToTargetDirection() noexcept = 0;
89
90     /**
91     * @brief метод, перемещающий животное в направлении direction
92     */
93     virtual void go() noexcept;
94
95 public:
96
97     /**
98     * @brief place — координаты животного на поле
99     */
100    Coordinates place;
101
102    /**
103    * @brief died — флаг; died = true, если животное умерло,
104    * died = false если животное живое
105    */
106    bool died;
107
108    virtual ~Animal() {}
109 };
110 #endif // ANIMAL_H

```

```

1 #include "animal.h"
2 #include <ctime>
3 #include <cstdlib>
4
5 void Animal::chooseEmptyDirection() noexcept
6 {
7     direction = field->whatIsEmpty(place.getV(), place.getH());
8     if (direction == Direction::NO_DIRECTION) {
9         has_moved = true;
10        direction = Direction::UP;

```

```

11     }
12 }
13
14 bool Animal::setDirection(Direction direction) noexcept
15 {
16     switch (direction)
17     {
18         case Direction::UP: {
19             if (field->isEmpty(place.getV() - 1, place.getH())) {
20                 this->direction = Direction::UP;
21                 return true;
22             }
23         }
24         case Direction::DOWN: {
25             if (field->isEmpty(place.getV() + 1, place.getH())) {
26                 this->direction = Direction::DOWN;
27                 return true;
28             }
29         }
30         case Direction::LEFT: {
31             if (field->isEmpty(place.getV(), place.getH() - 1)) {
32                 this->direction = Direction::LEFT;
33                 return true;
34             }
35         }
36         case Direction::RIGHT: {
37             if (field->isEmpty(place.getV(), place.getH() + 1)) {
38                 this->direction = Direction::RIGHT;
39                 return true;
40             }
41         }
42         default: {}
43     }
44     return false;
45 }
46
47 void Animal::chooseRandomDirection() noexcept
48 {
49     int flag = rand() % 4;
50     //TODO: будет понятнее, если в case тоже использовать enum
51     switch (flag) {
52         case 0: {
53             if (setDirection(Direction::UP) == false)
54                 chooseEmptyDirection();
55             break;
56         }
57         case 1: {
58             if (setDirection(Direction::RIGHT) == false)
59                 chooseEmptyDirection();
60             break;
61         }
62         case 2: {
63             if (setDirection(Direction::LEFT) == false)
64                 chooseEmptyDirection();
65             break;
66         }
67         case 3: {
68             if (setDirection(Direction::DOWN) == false)
69                 chooseEmptyDirection();
70             break;
71         }
72     }

```

```
73 | }  
74 |  
75 | void Animal::go() noexcept  
76 | {  
77 |     place.changeToDirection(direction);  
78 | }
```

```

1 #ifndef PREDATOR_H
2 #define PREDATOR_H
3 #include "modelapi.h"
4 #include "animal.h"
5
6 class Units;
7 class Prey;
8
9 /**
10  * @brief класс, реализующий хищника в модели
11  */
12 class Predator : public Animal
13 {
14     /**
15      * @brief DISTANCE_FOR_RESET_TARGET — дистанция, при которой
16      * ↪ жертва убегает от хищника
17      */
18     static constexpr double DISTANCE_FOR_RESET_TARGET = 2.1;
19 protected:
20     /**
21      * @brief PREDATOR_CREATE_ENERGY — энергия, необходимая для
22      * ↪ создания хищника
23      */
24     static const int PREDATOR_CREATE_ENERGY = 2;
25
26     /**
27      * @brief target — указатель на текущую цель
28      */
29     Prey* target;
30
31     /**
32      * @brief units_struct — указатель на класс с векторами хищников и
33      * ↪ жертв
34      */
35     Units* units_struct;
36
37     void directionFinding() noexcept;
38     void chooseToTargetDirection() noexcept;
39
40     /**
41      * @brief метод поиска жертвы на соседних 8 клетках;
42      * * в случае успеха записывает жертву в поле target
43      */
44     void findPrey() noexcept;
45
46     /**
47      * @brief метод, уничтожающий target — цель если(она есть)
48      */
49     void killPrey() noexcept;
50
51     /**
52      * @brief метод, создающий хищника на случайной соседней клетке;
53      * ↪ записывает его в вектор хищников
54      */
55     void createPredator() noexcept;
56 public:
57     /**
58      * @brief конструктор с параметрами; создает хищника на поле с
59      * ↪ указанными координатами

```

```

58     * @param v — координата по вертикали
59     * @param h — координата по горизонтали
60     * @param field_pointer — указатель на поле
61     * @param units_pointer — указатель на класс с векторами хищников и
    ↪ жертв
62     * @param time_of_life — время жизни хищника без еды
63     */
64     Predator(const int v, const int h, Field* field_pointer, Units*
    ↪ units_pointer, int time_of_life) noexcept;
65
66     /**
67     * @brief метод, передвигающий хищника
68     */
69     void movePredator() noexcept;
70 };
71
72 #endif // PREDATOR_H

```

```

1 #include "predator.h"
2 #include <vector>
3 #include <ctime>
4 #include <cmath>
5 #include <cstdlib>
6
7 void Predator::directionFinding() noexcept
8 {
9     if (target == nullptr){
10         chooseRandomDirection();
11     }
12     else {
13         double distance = place - target->place;
14         if (fabs(distance - DISTANCE_FOR_EAT) < DELTA) {
15             killPrey();
16         }
17         else chooseToTargetDirection();
18     }
19 }
20
21 void Predator::chooseToTargetDirection() noexcept
22 {
23     if ((place.getV() <= target->place.getV()) && (place.getH() <
    ↪ target->place.getH())) {
24         if (setDirection(Direction::RIGHT) == false)
25             if (setDirection(Direction::DOWN) == false)
    ↪ chooseRandomDirection();
26     }
27     if ((place.getV() < target->place.getV()) && (place.getH() >=
    ↪ target->place.getH())) {
28         if (setDirection(Direction::DOWN) == false)
29             if (setDirection(Direction::LEFT) == false)
    ↪ chooseRandomDirection();
30     }
31     if ((place.getV() > target->place.getV()) && (place.getH() <=
    ↪ target->place.getH())) {
32         if (setDirection(Direction::UP) == false)
33             if (setDirection(Direction::RIGHT) == false)
    ↪ chooseRandomDirection();
34     }
35     if ((place.getV() >= target->place.getV()) && (place.getH() >
    ↪ target->place.getH())) {
36         if (setDirection(Direction::LEFT) == false)
37             if (setDirection(Direction::UP) == false)

```



```

38     ↪ chooseRandomDirection();
39 }
40
41 void Predator::findPrey() noexcept
42 {
43     if (target != nullptr && (target->died == true ||
44         target->place - place > DISTANCE_FOR_RESET_TARGET))
45         ↪ target = nullptr;
46
47     double distance = 0;
48     for (Prey* prey: units_struct->preys) {
49         if (prey->died == false) {
50             distance = place - prey->place;
51             if (distance < DISTANCE_FOR_EAT + DELTA) {
52                 target = prey;
53                 break;
54             }
55             if (distance < DISTANCE_FOR_TARGET + DELTA) {
56                 target = prey;
57             }
58         }
59     }
60
61 void Predator::killPrey() noexcept
62 {
63     if (target->place.getV() < place.getV()) direction = Direction
64     ↪ ::UP;
65     else if (target->place.getV() > place.getV()) direction =
66     ↪ Direction::DOWN;
67     else if (target->place.getH() < place.getH()) direction =
68     ↪ Direction::LEFT;
69     else if (target->place.getH() > place.getH()) direction =
70     ↪ Direction::RIGHT;
71
72     target->died = true;
73
74     energy++;
75     life_time = -1;
76     target = nullptr;
77 }
78
79 void Predator::createPredator() noexcept
80 {
81     chooseRandomDirection();
82     switch (direction) {
83     case Direction::UP: {
84         new Predator(this->place.getV() - 1, this->place.getH(),
85         ↪ this->field,
86         ↪ this->units_struct, this
87         ↪ ->max_life_time);
88         break;
89     }
90     case Direction::RIGHT: {
91         new Predator(this->place.getV(), this->place.getH() + 1,
92         ↪ this->field,
93         ↪ this->units_struct, this
94         ↪ ->max_life_time);
95         break;
96     }
97 }

```

```

90     case Direction::DOWN: {
91         new Predator(this->place.getV() + 1, this->place.getH(),
92         ↪ this->field,
93                                     this->units_struct, this
94         ↪ ->max_life_time);
95         break;
96     }
97     case Direction::LEFT: {
98         new Predator(this->place.getV(), this->place.getH() - 1,
99         ↪ this->field,
100                                     this->units_struct, this
101         ↪ ->max_life_time);
102     }
103     default: {}
104 }
105
106 Predator::Predator(int v, int h, Field *field_pointer, Units *
107     ↪ units_pointer, int time_of_life) noexcept:
108     target(nullptr),
109     units_struct(units_pointer)
110 {
111     place.setV(v);
112     place.setH(h);
113     life_time = 0;
114     max_life_time = time_of_life;
115     energy = 0;
116     has_moved = false;
117     died = false;
118     field = field_pointer;
119     field->setPosition(place.getV(), place.getH(), Position::
120     ↪ PREDATOR);
121     direction = Direction::UP;
122     units_pointer->predators.push_back(this);
123 }
124
125 void Predator::movePredator() noexcept
126 {
127     findPrey();
128     directionFinding();
129     if (has_moved == false) {
130         field->setPosition(place.getV(), place.getH(), Position::
131         ↪ EMPTY);
132         go();
133         field->setPosition(place.getV(), place.getH(), Position::
134         ↪ PREDATOR);
135     }
136     else has_moved = false;
137
138     if (energy == PREDATOR_CREATE_ENERGY) {
139         createPredator();
140     }
141
142     life_time++;
143     if (life_time == max_life_time) {
144         field->setPosition(place.getV(), place.getH(), Position::
145         ↪ EMPTY);
146         died = true;
147     }

```

143 | }

```

1 #ifndef PREY_H
2 #define PREY_H
3 #include "animal.h"
4 #include "modelapi.h"
5
6 class Units;
7 class Grass;
8
9 /**
10  * @brief класс для реализации жертвы в модели
11  */
12 class Prey : public Animal
13 {
14 protected:
15     /**
16      * @brief PREY_CREATE_ENERGY — необходимая энергия для создания
17      * → жертвы
18      */
19     static const int PREY_CREATE_ENERGY = 2;
20
21     /**
22      * @brief warning — флаг, преследуется ли данная жертва
23      */
24     bool warning;
25
26     /**
27      * @brief dangerous_pred — координаты преследующего хищника
28      */
29     Coordinates dangerous_pred;
30
31     /**
32      * @brief units_struct — указатель на класс с векторами хищников и
33      * → жертв
34      */
35     Units* units_struct;
36
37     /**
38      * @brief target — указатель на текущую цель
39      */
40     Grass* target;
41
42     /**
43      * @brief метод поиска корма на соседних клетках;
44      * в случае успеха, записывает координаты в target
45      */
46     void findGrass();
47     void directionFinding() noexcept;
48     void chooseToTargetDirection() noexcept {} //пока нет травки
49
50     /**
51      * @brief метод, создающий жертву и записывающий ее в вектор
52      */
53     void createPrey();
54
55     /**
56      * @brief метод, проверяющий, не преследуется ли жертва
57      */
58     void isChase();
59
60 public:
61     /**

```

```

61     * @brief конструктор с параметрами
62     * @param v — координата по вертикали
63     * @param h — координата по горизонтали
64     * @param field_pointer — указатель на поле, где создается жертва
65     * @param units_pointer — указатель на класс с векторами хищников и
    ↪ жертв
66     */
67     Prey(const int v, const int h, Field* field_pointer, Units*
    ↪ units_pointer);
68
69     /**
70     * @brief метод, передвигающий жертву
71     */
72     void movePrey();
73
74     ~Prey() {}
75 };
76
77 #endif // PREY_H

```

```

1  #include "prey.h"
2  #include <ctime>
3  #include <cstdlib>
4
5  void Prey::findGrass()
6  {
7      if (target != nullptr && target->eaten == true) {
8          target = nullptr;
9      }
10
11      double distance = 0;
12      for (Grass* grass: units_struct->grass) {
13          if (grass->eaten == false) {
14              distance = place - grass->place;
15              if (distance < DISTANCE_FOR_EAT + DELTA) {
16                  target = grass;
17                  break;
18              }
19              if (distance < DISTANCE_FOR_TARGET + DELTA) {
20                  target = grass;
21              }
22          }
23      }
24  }
25
26  void Prey::directionFinding() noexcept
27  {
28      chooseRandomDirection();
29  }
30
31  //TODO: дублирует метод из predator, можно попытаться вынести общее в
    ↪ animal
32  void Prey::createPrey()
33  {
34      chooseRandomDirection();
35      switch (direction) {
36          case Direction::UP: {
37              new Prey(this->place.getV() - 1, this->place.getH(),
    ↪ this->field, this->units_struct);
38              break;
39          }
40          case Direction::RIGHT: {

```

```

41         new Prey(this->place.getV(), this->place.getH() + 1,
42         ↪ this->field, this->units_struct);
43         break;
44     }
45     case Direction::DOWN: {
46         new Prey(this->place.getV() + 1, this->place.getH(),
47         ↪ this->field, this->units_struct);
48         break;
49     }
50     case Direction::LEFT: {
51         new Prey(this->place.getV(), this->place.getH() - 1,
52         ↪ this->field, this->units_struct);
53     }
54     default: {}
55 }
56
57 this->energy = 0;
58 }
59
60 void Prey::isChase()
61 {
62     warning = false;
63     for (unsigned int i = 0; i < units_struct->predators.size(); ++
64     ↪ i) {
65         if (units_struct->predators[i] != nullptr) {
66             if (place - units_struct->predators[i]->place < 1.1) {
67                 warning = true;
68                 dangerous_pred = units_struct->predators[i]->place;
69                 break;
70             }
71         }
72     }
73 }
74
75 Prey::Prey(const int v, const int h, Field* field_pointer, Units *
76 ↪ units_pointer):
77     warning(false),
78     units_struct(units_pointer),
79     target(nullptr)
80 {
81     place.setV(v);
82     place.setH(h);
83     dangerous_pred.setV(-1);
84     dangerous_pred.setH(-1);
85     energy = 0;
86     life_time = 0;
87     has_moved = false;
88     died = false;
89     field = field_pointer;
90     field->setPosition(this->place.getV(), this->place.getH(),
91     ↪ Position::PREY);
92     direction = Direction::UP;
93     units_struct->preys.push_back(this);
94 }
95
96 void Prey::movePrey()
97 {
98     isChase();
99     directionFinding();
100     if (has_moved == false) {
101         field->setPosition(place.getV(), place.getH(), Position::
102         ↪ EMPTY);

```

```
96         go();
97         field->setPosition(place.getV(), place.getH(), Position::
↪ PREY);
98     }
99     else has_moved = false;
100     if (energy == PREY_CREATE_ENERGY) {
101         createPrey();
102     }
103 }
```

```

1 #ifndef COORDINATES_H
2 #define COORDINATES_H
3
4 /**
5  * @brief Direction — возможные значения направлений
6  */
7 enum class Direction
8 {
9     UP,
10    RIGHT,
11    DOWN,
12    LEFT,
13    NO_DIRECTION
14 };
15
16 /**
17  * @brief класс для представления координат объектов на поле
18  */
19 class Coordinates
20 {
21     /**
22      * @brief vertical — координата по вертикали
23      */
24     int vertical;
25
26     /**
27      * @brief horizontal — координата по горизонтали
28      */
29     int horizontal;
30
31 public:
32
33     /**
34      * @brief конструктор с параметрами; создает объект с заданными
35      * ↪ координатами
36      */
37     Coordinates(int vertical = 0, int horizontal = 0): vertical(
38         ↪ vertical), horizontal(horizontal) {}
39
40     /**
41      * @brief метод, изменяющий координаты в соответствие с переданным
42      * ↪ направлением
43      */
44     void changeToDirection(Direction);
45
46     /**
47      * @brief метод, устанавливающий координату по вертикали
48      */
49     void setV(int vertical) { this->vertical = vertical; }
50
51     /**
52      * @brief метод, устанавливающий координату по горизонтали
53      */
54     void setH(int horizontal) { this->horizontal = horizontal; }
55
56     /**
57      * @brief метод, возвращающий координату по вертикали
58      */
59     int getV() const { return vertical; }
60
61     /**
62      * @brief метод, возвращающий координату по горизонтали
63      */
64     int getH() const { return horizontal; }
65
66 private:
67     // Методы для изменения координат
68     void changeVertical(int vertical);
69     void changeHorizontal(int horizontal);
70 };
71
72 #endif

```



```

60     */
61     int getH() const { return horizontal; }
62
63     /**
64     * @brief Разность координат — расстояние между соответствующими
        ↪ точками на плоскости
65     */
66     double operator-(Coordinates &);
67     bool operator==(Coordinates a) const;
68     bool operator!=(Coordinates a) const;
69
70 };
71
72 #endif // COORDINATES_H

```

```

1 #include "coordinates.h"
2 #include <cmath>
3
4 void Coordinates::changeToDirection(Direction direction)
5 {
6     switch (direction) {
7         case Direction::UP:    { this->setV(this->getV() - 1);
        ↪ break; }
8         case Direction::RIGHT: { this->setH(this->getH() + 1);
        ↪ break; }
9         case Direction::LEFT:  { this->setH(this->getH() - 1);
        ↪ break; }
10        case Direction::DOWN:   { this->setV(this->getV() + 1);
        ↪ break; }
11        default: {}
12    }
13 }
14
15 double Coordinates::operator-(Coordinates &a)
16 {
17     double distance_between_points = 0;
18     distance_between_points = sqrt(pow(vertical - a.getV(), 2) +
        ↪ pow(horizontal - a.getH(), 2));
19
20     return distance_between_points;
21 }
22
23 bool Coordinates::operator==(Coordinates a) const
24 {
25     return (vertical == a.vertical && horizontal == a.horizontal);
26 }
27
28 bool Coordinates::operator!=(Coordinates a) const
29 {
30     return (vertical != a.vertical || horizontal != a.horizontal);
31 }

```

```

1 #ifndef BADBOUNDARY_H
2 #define BADBOUNDARY_H
3 #include <exception>
4
5 /**
6  * @brief классисключение—, генерируется при указании неверных индексов
7  */
8 class BadFieldBoundary : public std::exception
9 {
10     int vertical;
11     int horizontal;
12
13 public:
14     BadFieldBoundary(int v, int h): vertical(v), horizontal(h) {}
15     virtual const char *what() const throw()
16     {
17         const char *string = "Элемента_с_такими_индексами_не_
18         ↳ существует";
19         return string;
20     };
21
22 #endif // BADBOUNDARY_H

```

```

1 #ifndef BADNUM_H
2 #define BADNUM_H
3 #include <exception>
4
5 /**
6  * @brief классисключение—, генерируется при вводе числа, не
7  * ↳ принадлежащего
8  * указанному промежутку
9  */
10 class BadNum : public std::exception
11 {
12     int bad_number;
13
14     /**
15     * @brief min_boundary — нижняя граница промежутка
16     */
17     int min_boundary;
18
19     /**
20     * @brief max_boundary — верхняя граница промежутка
21     */
22     int max_boundary;
23
24 public:
25     BadNum(int bad_number, int min_boundary, int max_boundary):
26         bad_number(bad_number),
27         min_boundary(min_boundary),
28         max_boundary(max_boundary)
29     {}
30     virtual const char *what() const throw()
31     {
32         const char *string;
33         if (bad_number < min_boundary) {
34             string = "Введенное_значение_меньше_допустимого";
35         }
36         if (bad_number > max_boundary) {
37             string = "Введенное_значение_больше_допустимого";
38         }
39     }
40 }

```

```

37         }
38         return string;
39     }
40
41     /**
42      * @brief метод, возвращающий верхнюю границу промежутка
43      */
44     int getMaxBoundary() { return max_boundary; }
45
46     /**
47      * @brief метод, возвращающий нижнюю границу промежутка
48      */
49     int getMinBoundary() { return min_boundary; }
50 };
51
52 #endif // BADNUM_H

```

```

1 #ifndef BADFIELD_H
2 #define BADFIELD_H
3 #include <exception>
4 #include "field.h"
5
6 /**
7  * @brief классисключение—, генерируется при попытке создания поля со
8  * ↪ слишком
9  * большой маленькой() длиной
10 */
11 class BadFieldLength : public std::exception
12 {
13     int length;
14 public:
15     explicit BadFieldLength(int length): length(length) {}
16     virtual const char *what() const throw()
17     {
18         const char *string = "Невозможно_создать_поле_с_введенной_
19 ↪ длиной";
20         return string;
21     }
22     /**
23      * @brief метод, возвращающий минимальную длину поля
24      */
25     int getMinLength() { return Field::MIN_FIELD_SIZE; }
26
27     /**
28      * @brief метод, возвращающий максимальную длину поля
29      */
30     int getMaxLength() { return Field::MAX_FIELD_SIZE; }
31 };
32
33 /**
34  * @brief классисключение—, генерируется при попытке создания поля со
35  * ↪ слишком
36  * большой маленькой() высотой
37 */
38 class BadFieldHeight : public std::exception
39 {
40     int height;
41 public:
42     explicit BadFieldHeight(int height): height(height) {}
43     virtual const char *what() const throw()

```

```

43     {
44         const char *string = "Невозможно_создать_поле_с_введенной_
↪ высотой";
45         return string;
46     }
47
48     /**
49      * @brief метод, возвращающий минимальную высоту поля
50      */
51     int getMinHeight() { return Field::MIN_FIELD_SIZE; }
52
53     /**
54      * @brief метод, возвращающий максимальную высоту поля
55      */
56     int getMaxHeight() { return Field::MAX_FIELD_SIZE; }
57 };
58
59 #endif // BADFIELD_H

```

6.2 Консольное приложение

```
1 #include <iostream>
2 #include "model.h"
3 #include "consoleapp.h"
4
5 int main()
6 {
7     ConsoleApp console;
8     console.createConsole();
9
10    return 0;
11 }
```

```

1 #ifndef CONSOLEAPP_H
2 #define CONSOLEAPP_H
3 #include "consoledialog.h"
4 #include "consoledrawer.h"
5
6 /**
7  * @brief класс — консольное приложение
8  * создает модель, настройки и организует консольное взаимодействие с
9  * ↪ пользователем
10 */
11 class ConsoleApp
12 {
13     ConsoleDialog *dialog;
14     ConsoleDrawer *drawer;
15     Settings *settings;
16     Model *model;
17
18     /**
19      * @brief TIME_FOR_SLEEP — время мкс() между отрисовкой модели
20      */
21     const int TIME_FOR_SLEEP = 500000;
22
23     /**
24      * @brief метод, создающий модель и отрисовывающий информацию о
25      * ↪ ней в консоль
26      */
27     void startModel();
28
29 public:
30     ConsoleApp();
31     ConsoleApp(const ConsoleApp&) = delete;
32
33     /**
34      * @brief метод, создающий консольное приложение
35      */
36     void createConsole();
37
38     ~ConsoleApp();
39 };
40 #endif // CONSOLEAPP_H

```

```

1 #include "consoleapp.h"
2 #include "badfield.h"
3 #include "unistd.h"
4
5 void ConsoleApp::startModel()
6 {
7     this->model = new Model(this->settings);
8     this->drawer = new ConsoleDrawer(this->model);
9
10    this->drawer->showModel();
11    while (model->isEnd() == false) {
12        usleep(TIME_FOR_SLEEP);
13        this->model->movePredators();
14        this->model->movePreys();
15        this->model->remove();
16        this->drawer->showModel();
17    }
18    this->drawer->showResult();

```

```

19     delete this->drawer;
20     delete this->model;
21 }
22
23 ConsoleApp::ConsoleApp() :
24     drawer(nullptr),
25     model(nullptr)
26 {
27     this->settings = new Settings;
28     this->dialog = new ConsoleDialog(this->settings);
29 }
30
31 void ConsoleApp::createConsole()
32 {
33     int menuChoice = 0;
34     bool end = false;
35     while (!end) {
36         menuChoice = this->dialog->mainMenuPresentation();
37         switch (menuChoice) {
38             case 1: {
39                 this->startModel();
40                 break;
41             }
42
43             case 2: {
44                 dialog->settingsMenuPresentation();
45                 break;
46             }
47
48             case 0: {
49                 end = true;
50             }
51         }
52     }
53 }
54
55 ConsoleApp::~~ConsoleApp()
56 {
57     delete this->settings;
58     delete this->dialog;
59 }

```

```

1 #ifndef CONSOLEIALOG_H
2 #define CONSOLEIALOG_H
3 #include "settings.h"
4 #include "model.h"
5 #include "consoledrawer.h"
6 #include <string>
7
8 /**
9  * @brief класс, содержащий консольные меню для взаимодействия с
10  *   ↪ пользователем
11 */
12 class ConsoleDialog
13 {
14     std::string input_number;
15     Settings *settings;
16
17     /**
18     * @brief метод, выводющий в консоль меню изменения настроек поля
19     */
20     void changeFieldSize();
21
22     /**
23     * @brief метод, выводющий в консоль меню изменения времени жизни
24     *   ↪ животного без еды
25     */
26     void changeMovesWithoutMeal();
27
28     /**
29     * @brief метод, выводющий в консоль меню изменения числа хищников
30     */
31     void changeNumOfPredators();
32
33     /**
34     * @brief метод, выводющий в консоль меню изменения числа жертв
35     */
36     void changeNumOfPreys();
37
38     /**
39     * @brief метод, выводющий в консоль меню настроек
40     * @return int — выбранный пункт меню
41     */
42     int settingsPresentation();
43
44     /**
45     * @brief считывает из консоли число и возвращает его
46     */
47     int readInt();
48
49 public:
50     explicit ConsoleDialog(Settings *settings): settings(settings)
51     ↪ {}
52
53     /**
54     * @brief метод, выводющий в консоль главное меню
55     * @return int — выбранный пункт меню
56     */
57     int mainMenuPresentation();
58
59     /**
60     * @brief метод, обрабатывающий выбранный пункт в меню настроек
61     */
62     void settingsMenuPresentation();

```



```

60 };
61
62 #endif // CONSOLEIALOG_H

```

```

1 #include "consoledialog.h"
2 #include "badfield.h"
3 #include "badnum.h"
4 #include "badinput.h"
5 #include <iostream>
6
7 int ConsoleDialog::readInt()
8 {
9     std::getline(std::cin, this->input_number);
10    char *first_after_num;
11    int input = strtol(input_number.c_str(), &first_after_num, 10);
12    if ((input == 0 && input_number.size() > 1) || *first_after_num
    ↪ != '\0')
13    {
14        throw InputError();
15    }
16    return input;
17 }
18
19 void ConsoleDialog::changeFieldSize()
20 {
21     std::cout << "Максимальная_длина_и_высота_поля:_ " << Field::
    ↪ MAX_FIELD_SIZE << std::endl
22     << "Минимальная_длина_и_высота_поля:_ " << Field::
    ↪ MIN_FIELD_SIZE << std::endl;
23
24     std::cout << "Введите_высоту_поля:_ ";
25     int length = readInt();
26     settings->setFieldLength(length);
27
28     std::cout << "Введите_длину_поля:_ ";
29     int height = readInt();
30     settings->setFieldHeight(height);
31
32     std::cout << "Настройки_успешно_изменены!" << std::endl << std::
    ↪ endl;
33 }
34
35 void ConsoleDialog::changeMovesWithoutMeal()
36 {
37     std::cout << "Значение_должно_быть_в_пределах_от_"
38     << settings->getMinMovesWithoutMeal() << "до_"
39     << settings->getMaxMovesWithoutMeal() << std::endl
40     << "Введите_новое_время_жизни_хищника_в_(ходах):_ ";
41
42     int moves = readInt();
43     settings->setMovesWithoutMeal(moves);
44
45     std::cout << "Настройки_успешно_изменены!" << std::endl << std::
    ↪ endl;
46 }
47
48 void ConsoleDialog::changeNumOfPredators()
49 {
50     std::cout << "Значение_должно_быть_в_пределах_от_1_до_" <<
    ↪ settings->getMaxUnits() << std::endl
51     << "Введите_новое_число_хищников:_ ";
52

```

```

53     int number = readInt();
54     settings->setNumOfPredators(number);
55
56     std::cout << "Настройки_успешно_изменены!" << std::endl << std::
    ↪ endl;
57 }
58
59 void ConsoleDialog::changeNumOfPreys()
60 {
61     std::cout << "Значение_должно_быть_в_пределах_от_1_до_" <<
    ↪ settings->getMaxUnits() << std::endl
62         << "Введите_новое_число_жертв:_";
63
64     int number = readInt();
65     settings->setNumOfPreys(number);
66
67     std::cout << "Настройки_успешно_изменены!" << std::endl << std::
    ↪ endl;
68 }
69
70 void ConsoleDialog::settingsMenuPresentation()
71 {
72     int choice = -1;
73     while (choice != 0) {
74         choice = this->settingsPresentation();
75         try {
76             switch (choice) {
77                 case 1: {
78                     std::cout << std::endl;
79                     this->changeFieldSize();
80                     break;
81                 }
82                 case 2: {
83                     std::cout << std::endl;
84                     this->changeNumOfPredators();
85                     break;
86                 }
87                 case 3: {
88                     std::cout << std::endl;
89                     this->changeNumOfPreys();
90                     break;
91                 }
92                 case 4: {
93                     std::cout << std::endl;
94                     this->changeMovesWithoutMeal();
95                     break;
96                 }
97                 case 0: {
98                     std::cout << std::endl;
99                     break;
100                 }
101             }
102         }
103         catch (std::exception &e)
104         {
105             std::cout << e.what() << std::endl << std::endl;
106         }
107     }
108 }
109
110 int ConsoleDialog::mainMenuPresentation()
111 {

```

```

112     std::cout << "Модель_ХищникЖертва\"-\\" << std::endl;
113     std::cout << "1._Создать_новую_модель." << std::endl;
114     std::cout << "2._Настройки." << std::endl;
115     std::cout << "0._Выход." << std::endl;
116
117     int choice;
118     bool good_choice = false;
119     while (good_choice == false) {
120         std::cout << "Выберите_нужный_пункт_меню:_";
121         try {
122             choice = readInt();
123             if (choice < 3) {
124                 good_choice = true;
125                 return choice;
126             }
127             else std::cout << "Выбран_неверный_пункт_меню" << std
↵ ::endl;
128         }
129         catch (InputError) {
130             std::cout << "Выбран_неверный_пункт_меню" << std::endl;
131         }
132     }
133
134     return 0;
135 }
136
137 int ConsoleDialog::settingsPresentation()
138 {
139     std::cout << std::endl;
140     std::cout << "1._Именить_размеры_поля. .... Текущие_
↵ размеры_";
141     std::cout << settings->getFieldHeight() << "_x_" << settings->
↵ getFieldLength() << std::endl;
142     std::cout << "2._Изменить_количество_хищников. .... Текущее_
↵ число_";
143     std::cout << settings->getNumOfPredators() << std::endl;
144     std::cout << "3._Изменить_количество_жертв. .... Текущее_
↵ число_";
145     std::cout << settings->getNumOfPreys() << std::endl;
146     std::cout << "4._Изменить_время_жизни_хищника_без_еды. .... Текущее_
↵ время_";
147     std::cout << settings->getMovesWithoutMeal() << std::endl;
148     std::cout << "0._Назад" << std::endl;
149
150     bool good_choice = false;
151     int choice;
152     while (good_choice == false) {
153         std::cout << "Выберите_нужный_пункт_меню:_";
154         try {
155             choice = readInt();
156             if (choice < 5) {
157                 good_choice = true;
158                 return choice;
159             }
160             else std::cout << "Выбран_неверный_пункт_меню" << std
↵ ::endl;
161         }
162         catch (InputError) {
163             std::cout << "Выбран_неверный_пункт_меню" << std::endl;
164         }
165     }
166     return 0;

```

167 | }

```

1 #ifndef CONSOLEDRAWER_H
2 #define CONSOLEDRAWER_H
3 #include "model.h"
4
5 /**
6  * @brief класс, отрисовывающий в консоль информацию о модели
7  */
8 class ConsoleDrawer
9 {
10     Field *field;
11     Model *model;
12
13     /**
14      * @brief метод выводющий в консоль легенду
15      */
16     void drawLegend();
17
18     /**
19      * @brief метод, выводющий в консоль текущие день и время модели
20      */
21     void drawHead();
22
23     /**
24      * @brief метод, выводющий в консоль статистику:
25      * количество хищников и жертв на поле
26      */
27     void drawStatistics();
28
29     /**
30      * @brief метод, рисующий текущее состояние поля
31      */
32     void drawField();
33
34 public:
35     explicit ConsoleDrawer(Model *model): field(model->getField()),
36     ↪ model(model) {}
37
38     /**
39      * @brief метод, выводющий в консоль всю текущую информацию о
40     ↪ модели
41      */
42     void showModel();
43
44     /**
45      * @brief метод, выводющий в консоль результат победителей()
46      */
47     void showResult();
48 };
49 #endif // CONSOLEDRAWER_H

```

```

1 #include "consoledrawer.h"
2 #include <iostream>
3 #include <cstring>
4
5 void ConsoleDrawer::showModel()
6 {
7     std::cout << std::endl;
8     this->drawLegend();
9     this->drawHead();
10    this->drawStatistics();

```

```

11     this->drawField();
12 }
13
14 void ConsoleDrawer::showResult()
15 {
16     if (model->getPredatorsNum() == 0 && model->getPreysNum() > 0)
17         ↪ std::cout << "Жертвы_убежали_от_хищников!";
18     else if (model->getPredatorsNum() > 0 && model->getPreysNum()
19         ↪ == 0) std::cout << "Хищники_съели_всех_жертв!";
20     else if (model->getPredatorsNum() == 0 && model->getPreysNum()
21         ↪ == 0) std::cout << "Ничья!";
22
23     std::cout << std::endl << std::endl;
24 }
25
26 void ConsoleDrawer::drawHead()
27 {
28     int lenght_of_string = std::strlen("День_XX_Время_HH:MM");
29     int num_of_stars_left = (this->field->getLength() * 2 -
30         ↪ lenght_of_string) / 2;
31     int num_of_stars_right = this->field->getLength() * 2 -
32         ↪ lenght_of_string - num_of_stars_left;
33
34     for (int i = 0; i < num_of_stars_left; i++) {
35         std::cout << '*';
36     }
37
38     int day = this->model->getDay();
39     std::cout << "День_";
40     if (day < 10) std::cout << '0' << day << '_';
41     else std::cout << day << '_';
42
43     int time = this->model->getTime();
44     std::cout << "Время_";
45     if (time < 10) std::cout << '0' << time;
46     else std::cout << time;
47     std::cout << ':' << "00";
48
49     for (int i = 0; i < num_of_stars_right; i++) {
50         std::cout << '*';
51     }
52
53     std::cout << std::endl;
54 }
55
56 void ConsoleDrawer::drawStatistics()
57 {
58     int length_of_string = std::strlen("Хищники_XX_Жертвы_XX");
59     int num_of_stars_left = (this->field->getLength() * 2 -
60         ↪ length_of_string) / 2;
61     int num_of_stars_right = this->field->getLength() * 2 -
62         ↪ length_of_string - num_of_stars_left;
63
64     for (int i = 0; i < num_of_stars_left; i++)
65         std::cout << '*';
66
67     int predators = this->model->getPredatorsNum();
68     std::cout << "Хищники_";
69     if (predators >= 10) std::cout << predators;
70     else std::cout << '0' << predators;
71
72     for (int i = 0; i < num_of_stars_right; i++)
73         std::cout << '*';
74
75     std::cout << std::endl;
76 }

```

```

66     int preys = this->model->getPreysNum();
67     std::cout << "Жертвы";
68     if (preys >= 10) std::cout << preys;
69     else std::cout << '0' << preys;
70
71     for (int i = 0; i < num_of_stars_right; i++) {
72         std::cout << '*';
73     }
74
75     std::cout << std::endl;
76 }
77
78 void ConsoleDrawer::drawLegend()
79 {
80     std::cout << "X_хищники" << std::endl;
81     std::cout << "O_жертвы" << std::endl;
82 }
83
84 void ConsoleDrawer::drawField()
85 {
86     Position position;
87     for (int i = 0; i < this->field->getHeight(); i++) {
88         for (int j = 0; j < this->field->getLength(); j++) {
89             position = this->field->getPosition(i, j);
90             switch (position) {
91                 case Position::EMPTY: { std::cout << "."; break
92                 ↪ ; }
93                 case Position::PREDATOR: { std::cout << "X"; break
94                 ↪ ; }
95                 case Position::PREY: { std::cout << "O"; break
96                 ↪ ; }
97                 case Position::GRASS: { std::cout << "w"; break
98                 ↪ ; }
99             }
100             std::cout << std::endl;
101         }
102     }
103 }

```

```

1 #ifndef BADINPUT_H
2 #define BADINPUT_H
3 #include <exception>
4
5 /**
6  * @brief классисключение—, генерируется при неверном вводе
7  */
8 class InputError : public std::exception
9 {
10 public:
11     virtual const char *what() const throw()
12     {
13         const char *string = "Ошибка_ввода";
14         return string;
15     }
16 };
17
18 #endif // BADINPUT_H

```

6.3 Графическое приложение

```
1 #include "modelgui.h"
2
3 int main(int argc, char *argv[])
4 {
5     ModelGUI modelGUI(argc, argv);
6     modelGUI.startGUI();
7
8     return 0;
9 }
```

```
1 #ifndef MODELGUI_H
2 #define MODELGUI_H
3 #include "model.h"
4 #include "mainmenu.h"
5 #include <QApplication>
6
7 class ModelGUI
8 {
9     Settings* settings;
10
11     int argc;
12     char **argv;
13
14 public:
15     ModelGUI(int, char *[]);
16     int startGUI();
17
18     ~ModelGUI();
19 };
20
21 #endif // MODELGUI_H
```

```
1 #include "modelgui.h"
2
3 ModelGUI::ModelGUI(int argc, char *argv[]) :
4     argc(argc),
5     argv(argv)
6 {
7     settings = new Settings;
8 }
9
10 int ModelGUI::startGUI()
11 {
12     QApplication GUIapp(argc, argv);
13     MainMenu menu(nullptr, settings);
14     menu.show();
15
16     return GUIapp.exec();
17 }
18
19 ModelGUI::~~ModelGUI()
20 {
21     delete settings;
22 }
```



```

1 #ifndef MAINMENU_H
2 #define MAINMENU_H
3 #include "QtWidgets"
4 #include "exitwindow.h"
5 #include "settingswindow.h"
6 #include "settings.h"
7 #include "modelwindow.h"
8 #include "model.h"
9
10 /// это просто ужасно...
11 /// эту вещь хочется закинуть в класс, но компилятор не дает. Пишет, что
12   ↳ что-то там не литерал (
13   /// а вот так компилирует
14 static const QString button_style =
15     "QPushButton{"
16     "    border: 1px solid #324ab2;"
17     "    background: qlineargradient(x1: 0, y1: 1, x2: 0, y2: 0, stop: 0, stop: 1 #000000, stop: 1 #20155e);"
18     "    border-radius: 9px;"
19     "    color: #ffd7a8;"
20     "    font-size: 20px;"
21     "    font-weight: bold;"
22     "}"
23     "QPushButton:pressed{"
24     "    background: qlineargradient(x1: 0, y1: 1, x2: 0, y2: 0, stop: 0, stop: 1 #20155e, stop: 1 #000000);"
25     "}"
26     "QPushButton: hover{"
27     "    color: #ff7e00;"
28     "    border: 2.3px solid #f75e25;"
29     "}"
30 class MainMenu : public QWidget
31 {
32     Q_OBJECT
33
34     const QSize WINDOW_SIZE { 660, 540 };
35     const QSize BUTTON_SIZE { 180, 30 };
36
37     QPushButton* new_model_button;
38     QPushButton* settings_button;
39     QPushButton* exit_button;
40
41     Settings* settings;
42
43 public:
44     MainMenu(QWidget *parent, Settings*);
45
46 private slots:
47     void closeMenu();
48     void settingsMenu();
49     void createModel();
50 };
51
52 #endif // MAINMENU_H

```

```

1 #include "mainmenu.h"
2 #include "unistd.h"
3
4 MainMenu::MainMenu(QWidget* parent, Settings* settings) : QWidget(
    ↳ parent, Qt::WindowTitleHint)

```

```

5 {
6     this->setFixedSize(WINDOW_SIZE);
7     this->setWindowTitle("Хищникжертва-");
8     this->settings = settings;
9
10    QPixmap background(":/background2.jpg");
11    QPalette pal;
12    pal.setBrush(this->backgroundRole(), QBrush(background));
13    this->setPalette(pal);
14
15    new_model_button = new QPushButton("Новая_модель", this);
16    new_model_button->setStyleSheet(button_style);
17    new_model_button->resize(BUTTON_SIZE);
18    new_model_button->move(WINDOW_SIZE.width() - 200,
19                          WINDOW_SIZE.height() - 500);
20    connect(new_model_button, SIGNAL(clicked()), SLOT(createModel()
    ↪ ));
21
22    settings_button = new QPushButton("Настройки", this);
23    settings_button->setStyleSheet(button_style);
24    settings_button->resize(BUTTON_SIZE);
25    settings_button->move(WINDOW_SIZE.width() - 200,
26                          WINDOW_SIZE.height() - 450);
27    connect(settings_button, SIGNAL(clicked()), SLOT(settingsMenu()
    ↪ ));
28
29    exit_button = new QPushButton("Выход", this);
30    exit_button->setStyleSheet(button_style);
31    exit_button->resize(BUTTON_SIZE);
32    exit_button->move(WINDOW_SIZE.width() - 200,
33                     WINDOW_SIZE.height() - 400);
34    connect(exit_button, SIGNAL(clicked()), SLOT(closeMenu()));
35
36 }
37
38 void MainMenu::closeMenu()
39 {
40     ExitWindow* exit_menu = new ExitWindow(this);
41     exit_menu->exec();
42     delete exit_menu;
43 }
44
45 void MainMenu::settingsMenu()
46 {
47     SettingsWindow* settings_menu = new SettingsWindow(0, settings)
    ↪ ;
48     settings_menu->move(this->x(), this->y());
49     settings_menu->show();
50     this->close();
51 }
52
53 void MainMenu::createModel()
54 {
55     ModelWindow* model_window = new ModelWindow(0, settings);
56     model_window->move(this->x(), this->y());
57     model_window->show();
58     this->close();
59 }

```

```

1 #ifndef MODELWINDOW_H
2 #define MODELWINDOW_H
3 #include <QtWidgets>
4 #include <QPainter>
5 #include "model.h"
6 #include "mainmenu.h"
7 #include "fieldframe.h"
8 #include "statusframe.h"
9
10 class StatusFrame;
11
12 class ModelWindow : public QWidget
13 {
14     Q_OBJECT
15
16     Settings* settings;
17     Model* model;
18
19     const QSize WINDOW_SIZE { 660, 540 };
20     const QSize BUTTON_SIZE { 180, 30 };
21     const int TIME_FOR_MOVE = 500;
22
23     QPushButton* menu_button;
24     QPushButton* start_button;
25     QPushButton* generate_button;
26     QTimer* timer;
27
28     FieldFrame* field;
29     StatusFrame* status;
30
31     void endModel();
32
33 public:
34     ModelWindow(QWidget* parent, Settings* settings);
35     ~ModelWindow();
36
37 private slots:
38     void exitToMenu();
39     void startModel();
40     void moveModel();
41     void generateModel();
42 };
43
44 #endif // MODELWINDOW_H

```

```

1 #include "modelwindow.h"
2 #include "resultwindow.h"
3
4 ModelWindow::ModelWindow(QWidget* parent, Settings* settings) :
5     ↪ QWidget(parent, Qt::WindowTitleHint)
6 {
7     this->settings = settings;
8     this->setFixedSize(WINDOW_SIZE);
9     this->setWindowTitle("Хищникжертва-");
10
11     model = new Model(this->settings);
12
13     QPixmap background(":/settings_texture2.jpg");
14     QPalette pal;
15     pal.setBrush(this->backgroundRole(), QBrush(background));
16     this->setPalette(pal);

```

```

16 |
17 |     start_button = new QPushButton("Срайт", this);
18 |     start_button->setStyleSheet(button_style);
19 |     start_button->resize(BUTTON_SIZE);
20 |     start_button->move(WINDOW_SIZE.width() - 640, WINDOW_SIZE.
    ↪ height() - 515);
21 |     connect(start_button, SIGNAL(clicked()), SLOT(startModel()));
22 |
23 |     generate_button = new QPushButton("Сгенерировать", this);
24 |     generate_button->setStyleSheet(button_style);
25 |     generate_button->resize(BUTTON_SIZE);
26 |     generate_button->move(WINDOW_SIZE.width() - 420, WINDOW_SIZE.
    ↪ height() - 515);
27 |     connect(generate_button, SIGNAL(clicked()), SLOT(generateModel
    ↪ ()));
28 |
29 |     menu_button = new QPushButton("Выйти_в_меню", this);
30 |     menu_button->setStyleSheet(button_style);
31 |     menu_button->resize(BUTTON_SIZE);
32 |     menu_button->move(WINDOW_SIZE.width() - 200, WINDOW_SIZE.height
    ↪ () - 515);
33 |     connect(menu_button, SIGNAL(clicked()), SLOT(exitToMenu()));
34 |
35 |     timer = new QTimer(this);
36 |     connect(timer, SIGNAL(timeout()), this, SLOT(moveModel()));
37 |
38 |     field = new FieldFrame(this, model->getField());
39 |     field->show();
40 |
41 |     status = new StatusFrame(this, model);
42 |     status->show();
43 |     status->drawStatus();
44 | }
45 |
46 | ModelWindow::~ModelWindow()
47 | {
48 |     this->~QWidget();
49 |     delete model;
50 | }
51 |
52 | void ModelWindow::exitToMenu()
53 | {
54 |     MainMenu* menu_window = new MainMenu(0, settings);
55 |     menu_window->move(this->x(), this->y());
56 |     menu_window->show();
57 |     this->close();
58 |     delete model;
59 |     timer->stop();
60 | }
61 |
62 | void ModelWindow::startModel()
63 | {
64 |     start_button->setEnabled(false);
65 |     generate_button->setEnabled(false);
66 |     timer->start(TIME_FOR_MOVE);
67 | }
68 |
69 | void ModelWindow::moveModel()
70 | {
71 |     model->movePredators();
72 |     model->movePreys();
73 |     model->remove();

```

```

74     field->update();
75     status->drawStatus();
76     if (model->isEnd() == true) {
77         endModel();
78     }
79 }
80
81 void MainWindow::generateModel()
82 {
83     delete field;
84     delete status;
85     delete model;
86     model = new Model(settings);
87     field = new FieldFrame(this, model->getField());
88     status = new StatusFrame(this, model);
89     field->show();
90     status->show();
91     status->drawStatus();
92
93     start_button->setEnabled(true);
94 }
95
96 void MainWindow::endModel()
97 {
98     timer->stop();
99     if (model->getPredatorsNum() == 0) {
100         ResultWindow* result = new ResultWindow(this, "preys");
101         result->exec();
102         delete result;
103     }
104     if (model->getPreysNum() == 0) {
105         ResultWindow* result = new ResultWindow(this, "predators");
106         result->exec();
107         delete result;
108     }
109     generate_button->setEnabled(true);
110 }

```

```

1 #ifndef SETTINGSWINDOW_H
2 #define SETTINGSWINDOW_H
3 #include <QtWidgets>
4 #include "mainmenu.h"
5 #include "settings.h"
6
7 class SettingsWindow : public QWidget
8 {
9     Q_OBJECT
10
11     Settings* settings;
12
13     const QSize WINDOW_SIZE { 660, 540 };
14     const QSize BUTTON_SIZE { 200, 30 };
15
16     QPushButton* back_button;
17     QPushButton* save_button;
18
19     QLabel* field_length_label;
20     QLabel* field_height_label;
21     QLabel* predators_label;
22     QLabel* preys_label;
23     QLabel* moves_without_meal_label;
24     QLabel* success_label;
25     QTimer* timer_for_label;
26
27
28     QSpinBox* field_length;
29     QSpinBox* field_height;
30     QSpinBox* predators;
31     QSpinBox* preys;
32     QSpinBox* moves_without_meal;
33
34     QLabel* createLabel(QString text, int horizontal, int vertical,
35 ↪ bool invisiblity = false);
36     QSpinBox* createSpinBox(int min, int max, int horizontal, int
37 ↪ vertical);
38
39 public:
40     SettingsWindow(QWidget* parent, Settings* settings);
41
42 private slots:
43     void closeSettings();
44     void saveSettings();
45     void unlockSaveButton();
46 };
47
48 #endif // SETTINGSWINDOW_H

```

```

1 #include "settingswindow.h"
2 #include "field.h"
3
4 SettingsWindow::SettingsWindow(QWidget* parent, Settings *settings)
5 ↪ : QWidget(parent, Qt::WindowTitleHint)
6 {
7     this->setFixedSize(WINDOW_SIZE);
8     this->setWindowTitle("Настройки");
9     this->settings = settings;
10
11     QPixmap background(":/settings_texture2.jpg");
12     QPalette pal;

```

```

12 pal.setBrush(this→backgroundRole(), QBrush(background));
13 this→setPalette(pal);
14
15
16 field_length_label = createLabel("Длина_поля", WINDOW_SIZE.width
↪ () - 550, WINDOW_SIZE.height() - 480);
17 field_height_label = createLabel("Высота_поля", WINDOW_SIZE.
↪ width() - 550, WINDOW_SIZE.height() - 430);
18 predators_label = createLabel("Количество_хищников", WINDOW_SIZE.
↪ width() - 550, WINDOW_SIZE.height() - 380);
19 preys_label = createLabel("Количество_жертв", WINDOW_SIZE.width()
↪ - 550, WINDOW_SIZE.height() - 330);
20 moves_without_meal_label = createLabel("Время_жизни_хищника_без_
↪ еды",
21 WINDOW_SIZE.width() -
↪ 550, WINDOW_SIZE.height() - 280);
22 success_label = createLabel("Настройки_успешно_сохранены",
23 WINDOW_SIZE.width() - 500,
↪ WINDOW_SIZE.height() - 80, true);
24
25 field_length = createSpinBox(Field::MIN_FIELD_SIZE, Field::
↪ MAX_FIELD_SIZE,
26 WINDOW_SIZE.width() - 170,
↪ WINDOW_SIZE.height() - 480);
27 field_length→setValue(settings→getFieldLength());
28
29 field_height = createSpinBox(Field::MIN_FIELD_SIZE, Field::
↪ MAX_FIELD_SIZE,
30 WINDOW_SIZE.width() - 170,
↪ WINDOW_SIZE.height() - 430);
31 field_height→setValue(settings→getFieldHeight());
32
33 predators = createSpinBox(1, settings→getMaxUnits(),
↪ WINDOW_SIZE.width() - 170, WINDOW_SIZE.height() - 380);
34 predators→setValue(settings→getNumOfPredators());
35
36 preys = createSpinBox(1, settings→getMaxUnits(), WINDOW_SIZE.
↪ width() - 170, WINDOW_SIZE.height() - 330);
37 preys→setValue(settings→getNumOfPreys());
38
39 moves_without_meal = createSpinBox(settings→
↪ getMinMovesWithoutMeal(), settings→getMaxMovesWithoutMeal()
↪ ,
40 WINDOW_SIZE.width() - 170,
↪ WINDOW_SIZE.height() - 280);
41 moves_without_meal→setValue(settings→getMovesWithoutMeal());
42
43 back_button = new QPushButton("Назад", this);
44 back_button→setStyleSheet(button_style);
45 back_button→resize(BUTTON_SIZE);
46 back_button→move(WINDOW_SIZE.width() - 600, WINDOW_SIZE.height
↪ () - 130);
47 connect(back_button, SIGNAL(clicked()), SLOT(closeSettings()));
48
49 save_button = new QPushButton("Сохранить", this);
50 save_button→setStyleSheet(button_style);
51 save_button→resize(BUTTON_SIZE);
52 save_button→move(WINDOW_SIZE.width() - 260, WINDOW_SIZE.height
↪ () - 130);
53 connect(save_button, SIGNAL(clicked()), SLOT(saveSettings()));
54 }
55

```

```

56 QLabel* SettingsWindow::createLabel(QString text, int horizontal,
    ↪ int vertical, bool invisiblity)
57 {
58     QLabel* label = new QLabel(this);
59     label->setStyleSheet(
60         "color:~#122faa;"
61         "font-size:~18px;"
62         "font-weight:~bold;");
63     label->move(horizontal, vertical);
64     label->setText(text);
65     if (invisiblity == false) {
66         label->show();
67     }
68     else {
69         label->hide();
70     }
71
72     return label;
73 }
74
75 QSpinBox* SettingsWindow::createSpinBox(int min, int max, int
    ↪ horizontal, int vertical)
76 {
77     QSpinBox* spinBox = new QSpinBox(this);
78     spinBox->setRange(min, max);
79     spinBox->setStyleSheet(
80         "color:~#122faa;"
81         "font-size:~18px;"
82         "font-weight:~bold;");
83     spinBox->move(horizontal, vertical);
84     spinBox->show();
85
86     return spinBox;
87 }
88
89 void SettingsWindow::closeSettings()
90 {
91     MainMenu* menu_window = new MainMenu(0, settings);
92     menu_window->move(this->x(), this->y());
93     menu_window->show();
94     this->close();
95 }
96
97 void SettingsWindow::saveSettings()
98 {
99     save_button->setEnabled(false);
100     settings->setFieldHeight(field_height->value());
101     settings->setFieldLength(field_length->value());
102     predators->setMaximum(settings->getMaxUnits());
103     preys->setMaximum(settings->getMaxUnits());
104     settings->setMovesWithoutMeal(moves_without_meal->value());
105
106     try {
107         settings->setNumOfPredators(predators->value());
108     }
109     catch (std::exception&) {
110         settings->setNumOfPredators(settings->getMaxUnits());
111         predators->setValue(settings->getNumOfPredators());
112     }
113
114     try {
115         settings->setNumOfPreys(preys->value());

```



```

116     }
117     catch (std::exception&) {
118         settings->setNumOfPreys(settings->getMaxUnits());
119         preys->setValue(settings->getNumOfPreys());
120     }
121
122     success_label->show();
123     QTimer::singleShot(1000, this, SLOT(unlockSaveButton()));
124 }
125
126 void SettingsWindow::unlockSaveButton()
127 {
128     save_button->setEnabled(true);
129     success_label->hide();
130 }

```

```

1 #ifndef EXITWINDOW_H
2 #define EXITWINDOW_H
3 #include <QtWidgets>
4 #include "mainmenu.h"
5
6 class ExitWindow : public QDialog
7 {
8     Q_OBJECT
9
10     const QSize WINDOW_SIZE { 300, 90 };
11     const QSize BUTTON_SIZE { 120, 30 };
12
13     QPushButton* yes_button;
14     QPushButton* no_button;
15
16     QWidget* parent;
17     QLabel* exit_label;
18
19 public:
20     explicit ExitWindow(QWidget* parent);
21
22 private slots:
23     void closeApp();
24     void closeExitWindow();
25 };
26
27 #endif // EXITWINDOW_H

```

```

1 #include "exitwindow.h"
2
3 ExitWindow::ExitWindow(QWidget* parent) : QDialog(parent, Qt::
    ↳ WindowTitleHint)
4 {
5     this->parent = parent;
6     this->setFixedSize(WINDOW_SIZE);
7     this->setWindowTitle("Подтверждение_выхода");
8
9     QPixmap background(":/texture.jpg");
10    QPalette pal;
11    pal.setBrush(this->backgroundRole(), QBrush(background));
12    this->setPalette(pal);
13
14    exit_label = new QLabel(this);
15    exit_label->setStyleSheet(
16        "color:~#122faa;"
17        "font-size:~15px;"
18        "font-weight:~bold;");
19    exit_label->move(WINDOW_SIZE.width() - 283, WINDOW_SIZE.height
    ↳ () - 80);
20    exit_label->setText("Вы_действительно_хотите_выйти?");
21    exit_label->show();
22
23    yes_button = new QPushButton("Да", this);
24    yes_button->resize(BUTTON_SIZE);
25    yes_button->setStyleSheet(button_style);
26    yes_button->move(WINDOW_SIZE.width() - 275, WINDOW_SIZE.height
    ↳ () - 50);
27    connect(yes_button, SIGNAL(clicked()), SLOT(closeApp()));
28
29    no_button = new QPushButton("Нет", this);
30    no_button->resize(BUTTON_SIZE);

```

```

31     no_button->setStyleSheet(button_style);
32     no_button->move(WINDOW_SIZE.width() - 125, WINDOW_SIZE.height()
    ↪ - 50);
33     connect(no_button, SIGNAL(clicked()), SLOT(closeExitWindow()));
34 }
35
36 void ExitWindow::closeApp()
37 {
38     this->close();
39     this->parent->close();
40 }
41
42 void ExitWindow::closeExitWindow()
43 {
44     this->close();
45 }

```

```

1 #ifndef FIELDFRAME_H
2 #define FIELDFRAME_H
3 #include "field.h"
4 #include <QFrame>
5
6 class FieldFrame : public QFrame
7 {
8     Q_OBJECT
9
10    QSize field_size;
11    Field* field;
12    const QPoint FIELD_PLACE { 30, 80 };
13    static constexpr int LINE_WIDTH_DELTA = 1;
14
15    QWidget* parent;
16
17    int cell_size;
18    void createField(QPainter &painter);
19    void createUnits(QPainter &painter);
20    void paintEvent(QPaintEvent*);
21
22 public:
23     FieldFrame(QWidget *parent, Field*);
24     static constexpr int FIELD_SIDE = 450;
25 };
26
27 #endif // FIELDFRAME_H

```

```

1 #include "fieldframe.h"
2 #include <QPainter>
3 #include <QBrush>
4
5 FieldFrame::FieldFrame(QWidget *parent, Field* field) : QFrame(
6     ↪ parent)
7 {
8     this->field = field;
9     this->parent = parent;
10    cell_size = 0;
11    field_size.setHeight(FIELD_SIDE);
12    field_size.setWidth(FIELD_SIDE);
13
14    QPalette pal;
15    pal.setBrush(this->backgroundRole(), Qt::white);
16    this->setAutoFillBackground(true);
17    this->setPalette(pal);
18
19    cell_size = FIELD_SIDE / std::max(field->getLength(), field->
20    ↪ getHeight());
21 }
22
23 void FieldFrame::createField(QPainter &painter)
24 {
25     painter.setPen(QPen(Qt::black, 2, Qt::SolidLine));
26     field_size.setHeight(field->getHeight() * cell_size);
27     field_size.setWidth(field->getLength() * cell_size);
28
29     int vert_line_coordX = 0;
30     painter.drawLine(vert_line_coordX + LINE_WIDTH_DELTA, 0,
31     ↪ vert_line_coordX + LINE_WIDTH_DELTA, field_size.height());
32     for (int i = 0; i < field->getLength(); i++) {
33         vert_line_coordX += cell_size;

```

```

31     painter.drawLine(vert_line_coordX, 0, vert_line_coordX,
    ↪ field_size.height());
32 }
33
34     int horiz_line_coordY = 0;
35     painter.drawLine(0, horiz_line_coordY + LINE_WIDTH_DELTA,
    ↪ field_size.width(), horiz_line_coordY + LINE_WIDTH_DELTA);
36     for (int i = 0; i < field->getHeight(); i++) {
37         horiz_line_coordY += cell_size;
38         painter.drawLine(0, horiz_line_coordY, field_size.width(),
    ↪ horiz_line_coordY);
39     }
40     field_size.setWidth (vert_line_coordX + LINE_WIDTH_DELTA);
41     field_size.setHeight(horiz_line_coordY + LINE_WIDTH_DELTA);
42
43     this->setFixedSize(field_size);
44 }
45 }
46
47 void FieldFrame::createUnits(QPainter &painter)
48 {
49     QBrush brush;
50     brush.setStyle(Qt::SolidPattern);
51     for (int i = 0; i < field->getHeight(); i++) {
52         for (int j = 0; j < field->getLength(); j++) {
53             switch (field->getPosition(i, j)) {
54                 case Position::PREDATOR : {
55                     brush.setColor(Qt::red);
56                     painter.fillRect(j * cell_size +
    ↪ LINE_WIDTH_DELTA,
57                                     i * cell_size +
    ↪ LINE_WIDTH_DELTA,
58                                     cell_size - 2 *
    ↪ LINE_WIDTH_DELTA,
59                                     cell_size - 2 *
    ↪ LINE_WIDTH_DELTA,
60                                     brush);
61                     break;
62                 }
63                 case Position::PREY : {
64                     brush.setColor(Qt::blue);
65                     painter.fillRect(j * cell_size +
    ↪ LINE_WIDTH_DELTA,
66                                     i * cell_size +
    ↪ LINE_WIDTH_DELTA,
67                                     cell_size - 2 *
    ↪ LINE_WIDTH_DELTA,
68                                     cell_size - 2 *
    ↪ LINE_WIDTH_DELTA,
69                                     brush);
70                 }
71                 default : {}
72             }
73         }
74     }
75 }
76
77 void FieldFrame::paintEvent(QPaintEvent* event)
78 {
79     QFrame::paintEvent(event);
80     QPainter painter(this);
81     createField(painter);

```

```

82 |     createUnits ( painter );
83 |     this -> move ( FIELD_PLACE.x () + ( FieldFrame :: FIELD_SIDE - this ->
      |     ↪ width () ) / 2 ,
84 |                 FIELD_PLACE.y () + ( FieldFrame :: FIELD_SIDE - this ->
      |     ↪ height () ) / 2 );
85 | }

```

```

1 #ifndef STATUSFRAME_H
2 #define STATUSFRAME_H
3 #include <QtWidgets>
4 #include "model.h"
5 #include "modelwindow.h"
6
7 class ModelWindow;
8
9 class StatusFrame : public QFrame
10 {
11     Q_OBJECT
12
13     const QSize FRAME_SIZE { 150, 450 };
14     const QPoint PLACE { 500, 80 };
15     Model* model;
16
17     QLabel* predators;
18     QLabel* preys;
19     QLabel* day;
20     QLabel* time;
21
22     QString label_style =
23         "color: _#122faa;"
24         "font-size: _20px;"
25         "font-weight: _bold;";
26     QLabel* predators_number_label;
27     QLabel* preys_number_label;
28     QLabel* day_number_label;
29     QLabel* time_number_label;
30
31     void fillLabel(QLabel*, QString, int, int);
32     void paintEvent(QPaintEvent*);
33
34 public:
35     StatusFrame(QWidget* parent, Model* model);
36     void drawStatus();
37 };
38
39 #endif // STATUSFRAME_H

```

```

1 #include "statusframe.h"
2 #include <QPainter>
3
4 StatusFrame::StatusFrame(QWidget* parent, Model* model) : QFrame(
5     ↪ parent)
6 {
7     this->model = model;
8     this->setFixedSize(FRAME_SIZE);
9     this->move(PLACE);
10    this->setLineWidth(2);
11    this->setFrameStyle(QFrame::Box | QFrame::Plain);
12
13    predators_number_label = new QLabel(this);
14    predators_number_label->resize(50, 30);
15    preys_number_label = new QLabel(this);
16    preys_number_label->resize(50, 30);
17    day_number_label = new QLabel(this);
18    day_number_label->resize(50, 30);
19    time_number_label = new QLabel(this);
20    time_number_label->resize(65, 30);

```

```

21     predators = new QLabel(this);
22     fillLabel(predators, "Хищники", 10, 10);
23
24     preys = new QLabel(this);
25     fillLabel(preys, "Жертвы", 18, 110);
26
27     day = new QLabel(this);
28     fillLabel(day, "День", 48, 210);
29
30     time = new QLabel(this);
31     fillLabel(time, "Время", 43, 310);
32 }
33
34 void StatusFrame::fillLabel(QLabel* label, QString text, int
    ↪ horizontal, int vertical)
35 {
36     label->setStyleSheet(label_style);
37     label->move(horizontal, vertical);
38     label->setText(text);
39     label->show();
40 }
41
42 void StatusFrame::drawStatus()
43 {
44     QString output_string = QString::number(model->getPredatorsNum
    ↪ (), 10);
45     if (model->getPredatorsNum() < 10) {
46         fillLabel(predators_number_label, output_string, 65, 50);
47     }
48     else {
49         fillLabel(predators_number_label, output_string, 60, 50);
50     }
51
52     output_string = QString::number(model->getPreysNum(), 10);
53     if (model->getPreysNum() < 10) {
54         fillLabel(preys_number_label, output_string, 65, 150);
55     }
56     else {
57         fillLabel(preys_number_label, output_string, 60, 150);
58     }
59
60     output_string = QString::number(model->getDay(), 10);
61     if (model->getDay() < 10) {
62         fillLabel(day_number_label, output_string, 68, 250);
63     }
64     else {
65         fillLabel(day_number_label, output_string, 63, 250);
66     }
67
68     output_string = QString::number(model->getTime(), 10);
69     if (model->getTime() < 10) {
70         output_string = "0" + output_string + ":00";
71     }
72     else {
73         output_string = output_string + ":00";
74     }
75     fillLabel(time_number_label, output_string, 47, 350);
76 }
77
78 void StatusFrame::paintEvent(QPaintEvent* event)
79 {
80     QFrame::paintEvent(event);

```



```
81     QPainter painter(this);
82
83     QBrush brush;
84     brush.setStyle(Qt::SolidPattern);
85     brush.setColor(Qt::red);
86     painter.fillRect(115, 16, 25, 15, brush);
87     brush.setColor(Qt::blue);
88     painter.fillRect(110, 116, 25, 15, brush);
89 }
```

```

1 #ifndef RESULTWINDOW_H
2 #define RESULTWINDOW_H
3 #include <QtWidgets>
4 #include "mainmenu.h"
5
6 class ResultWindow : public QDialog
7 {
8     Q_OBJECT
9
10     const QSize WINDOW_SIZE { 300, 90 };
11     const QSize BUTTON_SIZE { 120, 30 };
12
13     QPushButton* ok_button;
14     QLabel* result_label;
15
16 public:
17     ResultWindow(QWidget* parent, QString winners);
18
19 private slots:
20     void closeWindow();
21 };
22
23 #endif // RESULTWINDOW_H

```

```

1 #include "resultwindow.h"
2
3 ResultWindow::ResultWindow(QWidget* parent, QString winners) :
4     QDialog(parent, Qt::WindowTitleHint)
5 {
6     this->setFixedSize(WINDOW_SIZE);
7     this->setWindowTitle("Результат");
8
9     QPixmap background(":/texture.jpg");
10    QPalette pal;
11    pal.setBrush(this->backgroundRole(), QBrush(background));
12    this->setPalette(pal);
13
14    ok_button = new QPushButton("OK", this);
15    ok_button->setStyleSheet(button_style);
16    ok_button->move(WINDOW_SIZE.width() / 2 - BUTTON_SIZE.width() /
17    ↪ 2, 50);
18    ok_button->resize(BUTTON_SIZE);
19    connect(ok_button, SIGNAL(clicked()), SLOT(closeWindow()));
20
21    result_label = new QLabel(this);
22    result_label->setStyleSheet(
23        "color:~#122faa;"
24        "font-size:~18px;"
25        "font-weight:~bold;");
26    if (winners == "predators") {
27        result_label->setText("Хищники_съели_всех_жертв");
28        result_label->move(20, 10);
29    }
30    else {
31        result_label->setText("Жертвы_убежали_от_хищников");
32        result_label->move(5, 10);
33    }
34    result_label->show();
35
36 void ResultWindow::closeWindow()

```

```

36 {
37     this->close();
38 }

```

6.4 Модульные тесты

```

1  #include <QString>
2  #include <QtTest>
3  #include <cmath>
4  #include "field.h"
5  #include "coordinates.h"
6  #include "badfield.h"
7  #include "badboundary.h"
8  #include "badnum.h"
9  #include "model.h"
10 #include "predator.h"
11
12 //TODO: сделать отдельный проект с функциональными тестами и добавить их
13 class ModelTest : public QObject
14 {
15     Q_OBJECT
16
17 public:
18     ModelTest();
19
20 private Q_SLOTS:
21     int doubleCompare(double a, double b);
22     void moveEnd(Units*);
23
24     void coordinatesTest();
25     void fieldTest();
26     void settingsTest();
27
28     void predatorMoveTest();
29     void predatorCreateTest();
30     void predatorMoveKillTest();
31     void predatorPriorityTest();
32     void predatorHungryTest();
33     void twoPredatorsTest();
34
35     void modelInitializeTest();
36     void debugTest();
37 };
38
39 ModelTest::ModelTest() {}
40
41 int ModelTest::doubleCompare(double a, double b)
42 {
43     if (fabs(a - b) < 10e-3) return 1;
44     return 0;
45 }
46
47 //TODO: думаю, этот тест нереально понять никому, кроме автора
48 void ModelTest::moveEnd(Units *units)
49 {
50     for (std::vector< Predator* >::iterator it = units->predators.
51 ↪ begin(); it != units->predators.end(); ++it) {
52         if ( (*it)->died == true ) {
53             delete (*it);
54             (*it) = nullptr;
55         }
56     }
57 }

```

```

54     }
55 }
56 units->predators.erase( std::remove(units->predators.begin(),
↪ units->predators.end(), nullptr),
                    units->predators.end() );
57
58
59 for (std::vector< Prey* >::iterator it = units->preys.begin();
↪ it != units->preys.end(); ++it) {
60     if ( (*it)->died == true ) {
61         delete (*it);
62         (*it) = nullptr;
63     }
64 }
65 units->preys.erase( std::remove(units->preys.begin(), units->
↪ preys.end(), nullptr),
                    units->preys.end() );
66 }
67
68
69 void ModelTest::coordinatesTest()
70 {
71     Coordinates A(0, 0), B(3, 4), C(-10, 10);
72     double dist1, dist2, dist3;
73     dist1 = B - A;
74     dist2 = C - A;
75     dist3 = B - C;
76     QVERIFY2(doubleCompare(dist1, 5), "wrong_distance_between_
↪ points");
77     QVERIFY2(doubleCompare(dist2, 14.142), "wrong_distance_between_
↪ points");
78     QVERIFY2(doubleCompare(dist3, 14.318), "wrong_distance_between_
↪ points");
79
80     QCOMPARE(A.getV(), 0);
81     A.setH(110000);
82     QCOMPARE(A.getH(), 110000);
83
84     A.setV(15); A.setH(15);
85     B.setV(15); B.setH(15);
86     QCOMPARE(A == B, true);
87     QCOMPARE(A != B, false);
88
89     B.setH(16);
90     QCOMPARE(A != B, true);
91     QCOMPARE(A == B, false);
92
93     Coordinates D;
94     QCOMPARE(D.getV(), 0);
95     QCOMPARE(D.getH(), 0);
96
97     D.changeToDirection(Direction::RIGHT);
98     QCOMPARE(D.getH(), 1);
99     D.changeToDirection(Direction::DOWN);
100    QCOMPARE(D.getV(), 1);
101    D.changeToDirection(Direction::LEFT);
102    QCOMPARE(D.getH(), 0);
103    D.changeToDirection(Direction::UP);
104    QCOMPARE(D.getV(), 0);
105 }
106
107 //TODO: очень длинный тест, надо разбить на несколько поменьше, на
↪ каждую тестируемую функциональность
108 void ModelTest::fieldTest()

```

```

109 {
110     Field field;
111
112     QCOMPARE(field.getLength(), 10);
113     QCOMPARE(field.getHeight(), 10);
114     QCOMPARE(field.isEmpty(0, 0), true);
115     QCOMPARE(field.isEmpty(4, 2), true);
116     QCOMPARE(field.isEmpty(10, 10), false);
117     QCOMPARE(field.isEmpty(-1, 0), false);
118     QCOMPARE(field.isEmpty(0, -2), false);
119     QCOMPARE(field.isEmpty(-1, 1), false);
120     QCOMPARE(field.isEmpty(10, 9), false);
121     QCOMPARE(field.getPosition(2, 5), Position::EMPTY);
122
123     field.setPosition(1, 4, Position::PREDATOR);
124     QCOMPARE(field.isEmpty(1, 4), false);
125     QCOMPARE(field.whatIsEmpty(4, 0), Direction::UP);
126     QCOMPARE(field.whatIsEmpty(0, 0), Direction::RIGHT);
127     QCOMPARE(field.whatIsEmpty(1, 4), Direction::UP);
128     QCOMPARE(field.whatIsEmpty(2, 4), Direction::RIGHT);
129     QCOMPARE(field.whatIsEmpty(0, 9), Direction::DOWN);
130     QVERIFY_EXCEPTION_THROWN(field.whatIsEmpty(10, 3),
131                               ↪ BadFieldBoundary);
132
133     field.setPosition(0, 1, Position::PREDATOR);
134     field.setPosition(1, 0, Position::PREY);
135     QCOMPARE(field.isEmpty(0, 0), true);
136     QCOMPARE(field.whatIsEmpty(0, 0), Direction::NO_DIRECTION);
137
138     field.setPosition(2, 5, Position::PREDATOR);
139     field.setPosition(3, 4, Position::PREY);
140     QVERIFY_EXCEPTION_THROWN(field.setPosition(-1, 0, Position::
141                               ↪ PREDATOR), BadFieldBoundary);
142     QVERIFY_EXCEPTION_THROWN(field.setPosition(10, 10, Position::
143                               ↪ PREY), BadFieldBoundary);
144     QCOMPARE(field.whatIsEmpty(2, 4), Direction::LEFT);
145 }
146
147 void ModelTest::settingsTest()
148 {
149     Settings settings;
150
151     QVERIFY_EXCEPTION_THROWN(settings.setFieldHeight(-1),
152                               ↪ BadFieldHeight);
153     QVERIFY_EXCEPTION_THROWN(settings.setFieldLength(-1),
154                               ↪ BadFieldLength);
155     QVERIFY_EXCEPTION_THROWN(settings.setNumOfPredators(-1), BadNum
156                               ↪ );
157 }
158
159 void ModelTest::predatorMoveTest()
160 {
161     Field field;
162     Units units;
163
164     Predator* tst_predator = new Predator(4, 4, &field, &units, 20)
165                               ↪ ;
166
167     field.setPosition(3, 4, Position::PREDATOR);
168     field.setPosition(4, 5, Position::PREDATOR);
169     field.setPosition(5, 4, Position::PREDATOR);

```

```

164     field.setPosition(4, 3, Position::PREDATOR);
165
166     tst_predator->movePredator();
167     QCOMPARE(tst_predator->place.getV(), 4);
168     QCOMPARE(tst_predator->place.getH(), 4);
169
170 }
171
172 void ModelTest::predatorMoveKillTest()
173 {
174     Field field(10, 10);
175     Units units;
176
177     new Prey(3, 3, &field, &units);
178     Predator* tst_predator = new Predator(4, 4, &field, &units, 20)
179     ↪ ;
180
181     units.predators[0]->movePredator();
182
183     QCOMPARE(tst_predator->place.getV(), 4);
184     QCOMPARE(tst_predator->place.getH(), 3);
185
186     units.predators[0]->movePredator();
187
188     QCOMPARE(tst_predator->place.getV(), 3);
189     QCOMPARE(tst_predator->place.getH(), 3);
190     moveEnd(&units);
191     QCOMPARE(units.preys.empty(), true);
192 }
193
194 void ModelTest::predatorCreateTest()
195 {
196     Field field(10, 10);
197     Units units;
198
199     new Prey(3, 3, &field, &units);
200     new Prey(2, 3, &field, &units);
201     Predator* tst_predator = new Predator(4, 4, &field, &units, 20)
202     ↪ ;
203
204     tst_predator->movePredator();
205     tst_predator->movePredator();
206     tst_predator->movePredator();
207
208     QCOMPARE(tst_predator->place.getV(), 2);
209     QCOMPARE(tst_predator->place.getH(), 3);
210     int pred_size = units.predators.size();
211     QCOMPARE(pred_size, 2);
212 }
213
214 void ModelTest::predatorHungryTest()
215 {
216     Field field;
217     Units units;
218
219     Predator* tst_predator = new Predator(4, 4, &field, &units, 20)
220     ↪ ;
221
222     for (int i = 0; i < 20; i++) {
223         tst_predator->movePredator();
224     }

```

```

223     moveEnd(&units);
224     int pred_size = units.predators.size();
225     QCOMPARE(pred_size, 0);
226
227 }
228
229 void ModelTest::twoPredatorsTest()
230 {
231     Field field;
232     Units units;
233
234     Predator* tst_predator1 = new Predator(4, 5, &field, &units,
↪ 20);
235     Predator* tst_predator2 = new Predator(2, 3, &field, &units,
↪ 20);
236     new Prey(3, 4, &field, &units);
237
238     tst_predator1->movePredator();
239     tst_predator2->movePredator();
240     moveEnd(&units);
241     tst_predator1->movePredator();
242     tst_predator2->movePredator();
243     moveEnd(&units);
244     QCOMPARE(units.preys.empty(), true);
245
246     field.setPosition(tst_predator1->place.getV(), tst_predator1->
↪ place.getH(), Position::EMPTY);
247     field.setPosition(tst_predator2->place.getV(), tst_predator2->
↪ place.getH(), Position::EMPTY);
248     tst_predator1->died = true;
249     tst_predator2->died = true;
250     moveEnd(&units);
251     QCOMPARE(units.predators.empty(), true);
252
253 }
254
255 void ModelTest::predatorPriorityTest()
256 {
257     Field field(10, 10);
258     Units units;
259
260     Predator* tst_predator = new Predator(4, 4, &field, &units, 20)
↪ ;
261     new Prey(3, 3, &field, &units);
262     new Prey(5, 4, &field, &units);
263
264     tst_predator->movePredator();
265
266     QCOMPARE(tst_predator->place.getV(), 5);
267     QCOMPARE(tst_predator->place.getH(), 4);
268 }
269
270 void ModelTest::modelInitializeTest()
271 {
272     Settings sett;
273     Model model(&sett);
274
275     QCOMPARE(model.getDay(), 0);
276     QCOMPARE(model.getTime(), 0);
277
278     model.movePreys();
279     model.movePredators();

```

```

280     model.remove();
281     QCOMPARE(model.getDay(), 0);
282     QCOMPARE(model.getTime(), 1);
283
284     model.movePredators();
285     model.movePreys();
286     model.remove();
287     QCOMPARE(model.getTime(), 2);
288
289     QCOMPARE(model.isEnd(), false);
290 }
291
292 //TODO: слишком общее название для такого длинного теста
293 void ModelTest::debugTest()
294 {
295     Field field(10, 10);
296     Units units;
297     new Predator(8, 9, &field, &units, 5);
298     new Predator(9, 9, &field, &units, 5);
299     new Predator(0, 0, &field, &units, 5);
300
301     new Prey(0, 1, &field, &units);
302     new Prey(0, 3, &field, &units);
303     new Prey(0, 2, &field, &units);
304     new Prey(0, 4, &field, &units);
305
306     int num_of_predators_moves = 0;
307     while (num_of_predators_moves < 5) {
308         for (unsigned int i = 0; i < units.predators.size(); i++)
309             ↪ units.predators[i] -> movePredator();
310         moveEnd(&units);
311         num_of_predators_moves++;
312     }
313     moveEnd(&units);
314     int final_vec_size = units.predators.size();
315     QCOMPARE(final_vec_size, 3);
316 }
317
318 QTEST_APPLESS_MAIN(ModelTest)
319
320 #include "tst_modeltest.moc"

```


Predator-prey

Создано системой Doxygen 1.8.8

Ср 1 Июн 2016 20:40:45

Оглавление

Глава 1

Иерархический список классов

1.1 Иерархия классов

Иерархия классов.

Animal	??
Predator	??
Prey	??
ConsoleApp	??
ConsoleDialog	??
ConsoleDrawer	??
Coordinates	??
exception	
BadFieldBoundary	??
BadFieldHeight	??
BadFieldLength	??
BadNum	??
InputError	??
Field	??
Grass	??
ModelAPI	??
Model	??
ModelGUI	??
QDialog	
ExitWindow	??
ResultWindow	??
QFrame	
FieldFrame	??
StatusFrame	??
QObject	
ModelTest	??
QWidget	
MainMenu	??
ModelWindow	??
SettingsWindow	??
Settings	??
Units	??

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Animal	класс, от которого наследуются хищники и жертвы	??
BadFieldBoundary	класс-исключение, генерируется при указании неверных индексов	??
BadFieldHeight	класс-исключение, генерируется при попытке создания поля со слишком большой (маленькой) высотой	??
BadFieldLength	класс-исключение, генерируется при попытке создания поля со слишком большой (маленькой) длиной	??
BadNum	класс-исключение, генерируется при вводе числа, не принадлежащего указанному промежутку	??
ConsoleApp	класс - консольное приложение создает модель, настройки и организует консольное взаимодействие с пользователем	??
ConsoleDialog	класс, содержащий консольные меню для взаимодействия с пользователем	??
ConsoleDrawer	класс, отрисовывающий в консоль информацию о модели	??
Coordinates	класс для представления координат объектов на поле	??
ExitWindow	??
Field	класс для представления поля в программе	??
FieldFrame	??
Grass	??
InputError	класс-исключение, генерируется при неверном вводе	??
MainMenu	??
Model	??
ModelAPI	класс, предоставляющий методы ядра	??
ModelGUI	??
ModelTest	??
ModelWindow	??
Predator	класс, реализующий хищника в модели	??

Prey		
	класс для реализации жертвы в модели	??
ResultWindow	??
Settings		
	класс, содержащий настройки модели	??
SettingsWindow	??
StatusFrame	??
Units		
	класс для содержания векторов хищников и жертв, а также корма для жертв . . .	??

Глава 3

Классы

3.1 Класс Animal

класс, от которого наследуются хищники и жертвы

```
#include <animal.h>
```

Граф наследования: Animal:

Граф связей класса Animal:

Открытые атрибуты

- [Coordinates place](#)
place - координаты животного на поле
- [bool died](#)
died - флаг; died = true, если животное умерло, died = false если животное живое

Защищенные члены

- [bool setDirection \(Direction\)](#) noexcept
метод устанавливает направление, если соответствующая клетка свободна
- [void chooseRandomDirection \(\)](#) noexcept
метод, выбирающий случайное направление, записывает его в direction.
- [virtual void directionFinding \(\)](#) noexcept=0
метод, выбирающий направление для следующего хода, записывает его в direction.
- [virtual void chooseToTargetDirection \(\)](#) noexcept=0
метод, выбирающий направление, в зависимости от положения цели
- [virtual void go \(\)](#) noexcept
метод, перемещающий животное в направлении direction.

Защищенные данные

- [int life_time](#)
life_time - счетчик ходов животного на поле
- [int max_life_time](#)
max_life_time - максимальное время жизни животного без еды
- [int energy](#)
energy - текущая энергия животного

- bool `has_moved`
`has_moved` - флаг; используется в случае, когда все четыре направления заблокированы
- Direction `direction`
`direction` - текущее направление животного
- Field * `field`
`field` - указатель на поле, где стоит животное

Статические защищенные данные

- static constexpr double `DISTANCE_FOR_EAT` = 1
`DISTANCE_FOR_KILL` - дистанция до жертвы, при которой можно ее съесть
- static constexpr double `DISTANCE_FOR_TARGET` = 1.4
`DISTANCE_FOR_TARGET` - дистанция для взятия жертвы в цель
- static constexpr double `DELTA` = 0.1
`DELTA` - константа, необходимая для сравнения чисел

3.1.1 Подробное описание

класс, от которого наследуются хищники и жертвы

3.1.2 Методы

3.1.2.1 bool Animal::setDirection (Direction direction) [protected], [noexcept]

метод устанавливает направление, если соответствующая клетка свободна

Возвращает

`true`, если удалось установить направление

Объявления и описания членов классов находятся в файлах:

- `/home/user/GIT/predator-prey/sources/Predator-prey/lib/animal.h`
- `/home/user/GIT/predator-prey/sources/Predator-prey/lib/animal.cpp`

3.2 Класс BadFieldBoundary

класс-исключение, генерируется при указании неверных индексов

```
#include <badboundary.h>
```

Граф наследования: BadFieldBoundary:

Граф связей класса BadFieldBoundary:

Открытые члены

- BadFieldBoundary (int v, int h)
- virtual const char * what () const throw ()

3.2.1 Подробное описание

класс-исключение, генерируется при указании неверных индексов

Объявления и описания членов класса находятся в файле:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/badboundary.h

3.3 Класс BadFieldHeight

класс-исключение, генерируется при попытке создания поля со слишком большой (маленькой) высотой

```
#include <badfield.h>
```

Граф наследования:BadFieldHeight:

Граф связей класса BadFieldHeight:

Открытые члены

- BadFieldHeight (int height)
- virtual const char * what () const throw ()
- int [getMinHeight](#) ()
метод, возвращающий минимальную высоту поля
- int [getMaxHeight](#) ()
метод, возвращающий максимальную высоту поля

3.3.1 Подробное описание

класс-исключение, генерируется при попытке создания поля со слишком большой (маленькой) высотой

Объявления и описания членов класса находятся в файле:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/badfield.h

3.4 Класс BadFieldLength

класс-исключение, генерируется при попытке создания поля со слишком большой (маленькой) длиной

```
#include <badfield.h>
```

Граф наследования:BadFieldLength:

Граф связей класса BadFieldLength:

Открытые члены

- BadFieldLength (int length)
- virtual const char * what () const throw ()
- int [getMinLength](#) ()
метод, возвращающий минимальную длину поля
- int [getMaxLength](#) ()
метод, возвращающий максимальную длину поля

3.4.1 Подробное описание

класс-исключение, генерируется при попытке создания поля со слишком большой (маленькой) длиной

Объявления и описания членов класса находятся в файле:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/badfield.h

3.5 Класс BadNum

класс-исключение, генерируется при вводе числа, не принадлежащего указанному промежутку

```
#include <badnum.h>
```

Граф наследования:BadNum:

Граф связей класса BadNum:

Открытые члены

- BadNum (int bad_number, int min_boundary, int max_boundary)
- virtual const char * what () const throw ()
- int [getMaxBoundary](#) ()
метод, возвращающий верхнюю границу промежутка
- int [getMinBoundary](#) ()
метод, возвращающий нижнюю границу промежутка

3.5.1 Подробное описание

класс-исключение, генерируется при вводе числа, не принадлежащего указанному промежутку

Объявления и описания членов класса находятся в файле:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/badnum.h

3.6 Класс ConsoleApp

класс - консольное приложение создает модель, настройки и организует консольное взаимодействие с пользователем

```
#include <consoleapp.h>
```

Открытые члены

- ConsoleApp (const [ConsoleApp](#) &)=delete
- void [createConsole](#) ()
метод, создающий консольное приложение

3.6.1 Подробное описание

класс - консольное приложение создает модель, настройки и организует консольное взаимодействие с пользователем

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/consoleApp/consoleapp.h
- /home/user/GIT/predator-prey/sources/Predator-prey/consoleApp/consoleapp.cpp

3.7 Класс ConsoleDialog

класс, содержащий консольные меню для взаимодействия с пользователем

```
#include <consoledialog.h>
```

Открытые члены

- ConsoleDialog ([Settings](#) *settings)
- int [mainMenuPresentation](#) ()
метод, выводящий в консоль главное меню
- void [settingsMenuPresentation](#) ()
метод, обрабатывающий выбранный пункт в меню настроек

3.7.1 Подробное описание

класс, содержащий консольные меню для взаимодействия с пользователем

3.7.2 Методы

3.7.2.1 int ConsoleDialog::mainMenuPresentation ()

метод, выводящий в консоль главное меню

Возвращает

int - выбранный пункт меню

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/consoleApp/consoledialog.h
- /home/user/GIT/predator-prey/sources/Predator-prey/consoleApp/consoledialog.cpp

3.8 Класс ConsoleDrawer

класс, отрисовывающий в консоль информацию о модели

```
#include <consoledrawer.h>
```

Открытые члены

- ConsoleDrawer ([Model](#) *model)
- void [showModel](#) ()
метод, выводящий в консоль всю текущую информацию о модели
- void [showResult](#) ()
метод, выводящий в консоль результат (победителей)

3.8.1 Подробное описание

класс, отрисовывающий в консоль информацию о модели

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/consoleApp/consolodrawer.h
- /home/user/GIT/predator-prey/sources/Predator-prey/consoleApp/consolodrawer.cpp

3.9 Класс Coordinates

класс для представления координат объектов на поле

```
#include <coordinates.h>
```

Открытые члены

- **Coordinates** (int vertical=0, int horizontal=0)
конструктор с параметрами; создает объект с заданными координатами
- void **changeToDirection** (Direction)
метод, изменяющий координаты в соответствии с переданным направлением
- void **setV** (int vertical)
метод, устанавливающий координату по вертикали
- void **setH** (int horizontal)
метод, устанавливающий координату по горизонтали
- int **getV** () const
метод, возвращающий координату по вертикали
- int **getH** () const
метод, возвращающий координату по горизонтали
- double **operator-** (**Coordinates** &)
Разность координат - расстояние между соответствующими точками на плоскости
- bool operator== (**Coordinates** a) const
- bool operator!= (**Coordinates** a) const

3.9.1 Подробное описание

класс для представления координат объектов на поле

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/coordinates.h
- /home/user/GIT/predator-prey/sources/Predator-prey/lib/coordinates.cpp

3.10 Класс ExitWindow

Граф наследования:ExitWindow:

Граф связей класса ExitWindow:

Открытые члены

- ExitWindow (QWidget *parent)

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/exitwindow.h
- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/exitwindow.cpp

3.11 Класс Field

класс для представления поля в программе

```
#include <field.h>
```

Открытые члены

- **Field** (int height=10, int length=10)
конструктор с параметрами, создает поле указанных размеров
- bool **isEmpty** (int v, int h) const
метод, позволяющий узнать, является ли клетка с данными координатами пустым
- void **setPosition** (int v, int h, Position)
метод, позволяющий установить на клетку с данными координатами заданный символ
- Direction **whatIsEmpty** (int v, int h) const
метод, возвращающий свободное направление хода для заданной клетки
- Position **getPosition** (int v, int h) const
метод, возвращающий значение клетки с заданными координатами
- int **getLength** () const
метод, возвращающий длину поля в клетках
- int **getHeight** () const
метод, возвращающий высоту поля в клетках

Статические открытые данные

- static constexpr int **MAX_FIELD_SIZE** = 30
MAX_FIELD_SIZE - максимальная длина и высота поля
- static constexpr int **MIN_FIELD_SIZE** = 10
MIN_FIELD_SIZE - минимальная длина и высота поля

3.11.1 Подробное описание

класс для представления поля в программе

3.11.2 Конструктор(ы)

3.11.2.1 Field::Field (int height = 10, int length = 10)

конструктор с параметрами, создает поле указанных размеров

Аргументы

height	- высота поля
length	- длина поля

3.11.3 Методы

3.11.3.1 int Field::getHeight () const [inline]

метод, возвращающий высоту поля в клетках

Возвращает

высота поля

3.11.3.2 int Field::getLength () const [inline]

метод, возвращающий длину поля в клетках

Возвращает

длина поля

3.11.3.3 Position Field::getPosition (int v, int h) const

метод, возвращающий значение клетки с заданными координатами

Аргументы

v	- координата по вертикали
h	- координата по горизонтали

Возвращает

символ - значение

3.11.3.4 bool Field::isEmpty (int v, int h) const

метод, позволяющий узнать, является ли клетка с данными координатами пустым

Аргументы

v	- координата по вертикали
h	- координата по горизонтали

Возвращает

возвращает true, если клетка свободна и false, если клетка занята

3.11.3.5 void Field::setPosition (int v, int h, Position pos)

метод, позволяющий установить на клетку с данными координатами заданный символ

Аргументы

v	- координата по вертикали
h	- координата по горизонтали
Position	- позиция, которую надо установить

3.11.3.6 Direction Field::whatIsEmpty (int v, int h) const

метод, возвращающий свободное направление хода для заданной клетки

Аргументы

v	- координата клетки по вертикали
h	- координата клетки по горизонтали

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/field.h
- /home/user/GIT/predator-prey/sources/Predator-prey/lib/field.cpp

3.12 Класс FieldFrame

Граф наследования:FieldFrame:

Граф связей класса FieldFrame:

Открытые члены

- FieldFrame (QWidget *parent, [Field](#) *)

Статические открытые данные

- static constexpr int FIELD_SIDE = 450

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/fieldframe.h
- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/fieldframe.cpp

3.13 Класс Grass

Граф связей класса Grass:

Открытые члены

- Grass (const int v, const int h, [Units](#) *units)

Открытые атрибуты

- [Coordinates](#) place
- bool eaten

Объявления и описания членов классов находятся в файлах:

- `/home/user/GIT/predator-prey/sources/Predator-prey/lib/grass.h`
- `/home/user/GIT/predator-prey/sources/Predator-prey/lib/grass.cpp`

3.14 Класс `InputError`

класс-исключение, генерируется при неверном вводе

`#include <badinput.h>`

Граф наследования: `InputError`:

Граф связей класса `InputError`:

Открытые члены

- `virtual const char * what () const throw ()`

3.14.1 Подробное описание

класс-исключение, генерируется при неверном вводе

Объявления и описания членов класса находятся в файле:

- `/home/user/GIT/predator-prey/sources/Predator-prey/consoleApp/badinput.h`

3.15 Класс `MainMenu`

Граф наследования: `MainMenu`:

Граф связей класса `MainMenu`:

Открытые члены

- `MainMenu (QWidget *parent, Settings *)`

Объявления и описания членов классов находятся в файлах:

- `/home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/mainmenu.h`
- `/home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/mainmenu.cpp`

3.16 Класс `Model`

Граф наследования: `Model`:

Граф связей класса `Model`:

Открытые члены

- `Model (Settings *settings) noexcept`
- `Field * getField () noexcept`
метод, возвращающий указатель на поле модели
- `int getTime () const noexcept`
метод, возвращающий текущее время

- int `getDay` () const noexcept
метод, возвращающий текущий день
- unsigned int `getPredatorsNum` () const noexcept
метод, возвращающий количество хищников на поле
- unsigned int `getPreysNum` () const noexcept
метод, возвращающий количество жертв на поле
- void `movePreys` () noexcept
метод, передвигающий жертв
- void `movePredators` () noexcept
метод, передвигающий хищников
- bool `isEnd` () const noexcept
метод, проверяющий, не исчезли ли хищники или жертвы
- void `createPredators` () noexcept
метод, создающий хищников
- void `createPreys` () noexcept
метод, создающий жертв
- void `removePredators` () noexcept
метод, удаляющий умерших хищников после хода
- void `removePreys` () noexcept
метод, удаляющий умерших жертв после хода
- void `remove` () noexcept
- void `saveModel` ()
- void `loadModel` ()

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/model.h
- /home/user/GIT/predator-prey/sources/Predator-prey/lib/model.cpp

3.17 Класс ModelAPI

класс, предоставляющий методы ядра

```
#include <modelapi.h>
```

Граф наследования: ModelAPI:

Открытые члены

- virtual `Field * getField` () noexcept=0
метод, возвращающий указатель на поле модели
- virtual int `getTime` () const noexcept=0
метод, возвращающий текущее время
- virtual int `getDay` () const noexcept=0
метод, возвращающий текущий день
- virtual unsigned int `getPredatorsNum` () const noexcept=0
метод, возвращающий количество хищников на поле
- virtual unsigned int `getPreysNum` () const noexcept=0
метод, возвращающий количество жертв на поле
- virtual bool `isEnd` () const noexcept=0
метод, проверяющий, не исчезли ли хищники или жертвы
- virtual void `createPredators` () noexcept=0

- метод, создающий хищников
- virtual void `createPreys` () noexcept=0
- метод, создающий жертв
- virtual void `removePredators` () noexcept=0
- метод, удаляющий умерших хищников после хода
- virtual void `removePreys` () noexcept=0
- метод, удаляющий умерших жертв после хода
- virtual void `movePreys` () noexcept=0
- метод, передвигающий жертв
- virtual void `movePredators` () noexcept=0
- метод, передвигающий хищников
- void `saveModel` ()
- void `loadModel` ()

3.17.1 Подробное описание

класс, предоставляющий методы ядра

Объявления и описания членов класса находятся в файле:

- `/home/user/GIT/predator-prey/sources/Predator-prey/lib/modelapi.h`

3.18 Класс ModelGUI

Открытые члены

- `ModelGUI` (int, char *[])
- `int startGUI` ()

Объявления и описания членов классов находятся в файлах:

- `/home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/modelgui.h`
- `/home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/modelgui.cpp`

3.19 Класс ModelTest

Граф наследования:ModelTest:

Граф связей класса ModelTest:

Объявления и описания членов класса находятся в файле:

- `/home/user/GIT/predator-prey/sources/Predator-prey/tests/tst_modeltest.cpp`

3.20 Класс ModelWindow

Граф наследования:ModelWindow:

Граф связей класса ModelWindow:

Открытые члены

- `ModelWindow (QWidget *parent, Settings *settings)`

Объявления и описания членов классов находятся в файлах:

- `/home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/modelwindow.h`
- `/home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/modelwindow.cpp`

3.21 Класс Predator

класс, реализующий хищника в модели

```
#include <predator.h>
```

Граф наследования: Predator:

Граф связей класса Predator:

Открытые члены

- [Predator](#) (const int v, const int h, [Field](#) *field_pointer, [Units](#) *units_pointer, int time_of_life)
noexcept
конструктор с параметрами; создает хищника на поле с указанными координатами
- void [movePredator](#) () noexcept
метод, передвигающий хищника

Защищенные члены

- void [directionFinding](#) () noexcept
метод, выбирающий направление для следующего хода, записывает его в `direction`.
- void [chooseToTargetDirection](#) () noexcept
метод, выбирающий направление, в зависимости от положения цели
- void [findPrey](#) () noexcept
метод поиска жертвы на соседних 8 клетках; в случае успеха записывает жертву в поле `target`.
- void [killPrey](#) () noexcept
метод, уничтожающий `target` - цель (если она есть)
- void [createPredator](#) () noexcept
метод, создающий хищника на случайной соседней клетке; записывает его в вектор хищников

Защищенные данные

- [Prey](#) * [target](#)
`target` - указатель на текущую цель
- [Units](#) * [units_struct](#)
`units_struct` - указатель на класс с векторами хищников и жертв

Статические защищенные данные

- static const int [PREDATOR_CREATE_ENERGY](#) = 2
`PREDATOR_CREATE_ENERGY` - энергия, необходимая для создания хищника

Дополнительные унаследованные члены

3.21.1 Подробное описание

класс, реализующий хищника в модели

3.21.2 Конструктор(ы)

3.21.2.1 `Predator::Predator (const int v, const int h, Field * field_pointer, Units * units_pointer, int time_of_life) [noexcept]`

конструктор с параметрами; создает хищника на поле с указанными координатами

Аргументы

<code>v</code>	- координата по вертикали
<code>h</code>	- координата по горизонтали
<code>field_pointer</code>	- указатель на поле
<code>units_pointer</code>	- указатель на класс с векторами хищников и жертв
<code>time_of_life</code>	- время жизни хищника без еды

Объявления и описания членов классов находятся в файлах:

- `/home/user/GIT/predator-prey/sources/Predator-prey/lib/predator.h`
- `/home/user/GIT/predator-prey/sources/Predator-prey/lib/predator.cpp`

3.22 Класс Prey

класс для реализации жертвы в модели

`#include <prey.h>`

Граф наследования: `Prey`:

Граф связей класса `Prey`:

Открытые члены

- `Prey (const int v, const int h, Field *field_pointer, Units *units_pointer)`
конструктор с параметрами
- `void movePrey ()`
метод, передвигающий жертву

Защищенные члены

- `void findGrass ()`
метод поиска корма на соседних клетках; в случае успеха, записывает координаты в `target`.
- `void directionFinding () noexcept`
метод, выбирающий направление для следующего хода, записывает его в `direction`.
- `void chooseToTargetDirection () noexcept`
метод, выбирающий направление, в зависимости от положения цели
- `void createPrey ()`
метод, создающий жертву и записывающий ее в вектор
- `void isChase ()`
метод, проверяющий, не преследуется ли жертва

Защищенные данные

- bool `warning`
warning - флаг, преследуется ли данная жертва
- `Coordinates dangerous_pred`
dangerous_pred - координаты преследующего хищника
- `Units * units_struct`
units_struct - указатель на класс с векторами хищников и жертв
- `Grass * target`
target - указатель на текущую цель

Статические защищенные данные

- static const int `PREY_CREATE_ENERGY` = 2
PREY_CREATE_ENERGY - необходимая энергия для создания жертвы

Дополнительные унаследованные члены

3.22.1 Подробное описание

класс для реализации жертвы в модели

3.22.2 Конструктор(ы)

3.22.2.1 Prey::Prey (const int v, const int h, Field * field_pointer, Units * units_pointer)

конструктор с параметрами

Аргументы

v	- координата по вертикали
h	- координата по горизонтали
field_pointer	- указатель на поле, где создается жертва
units_pointer	- указатель на класс с векторами хищников и жертв

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/prey.h
- /home/user/GIT/predator-prey/sources/Predator-prey/lib/prey.cpp

3.23 Класс ResultWindow

Граф наследования:ResultWindow:

Граф связей класса ResultWindow:

Открытые члены

- ResultWindow (QWidget *parent, QString winners)

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/resultwindow.h
- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/resultwindow.cpp

3.24 Класс Settings

класс, содержащий настройки модели

```
#include <settings.h>
```

Открытые члены

- int [getFieldLength](#) () const
методы, возвращающие информацию о текущих настройках
- int [getFieldHeight](#) () const
- int [getNumOfPreys](#) () const
- int [getNumOfPredators](#) () const
- int [getMovesWithoutMeal](#) () const
- int [getMinMovesWithoutMeal](#) () const
- int [getMaxMovesWithoutMeal](#) () const
- int [getMaxUnits](#) () const
- void [setFieldLength](#) (const int)
методы, устанавливающие новые настройки; при необходимости генерируют исключения
- void [setFieldHeight](#) (const int)
- void [setNumOfPredators](#) (const int)
- void [setNumOfPreys](#) (const int)
- void [setMovesWithoutMeal](#) (const int)

3.24.1 Подробное описание

класс, содержащий настройки модели

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/settings.h
- /home/user/GIT/predator-prey/sources/Predator-prey/lib/settings.cpp

3.25 Класс SettingsWindow

Граф наследования:SettingsWindow:

Граф связей класса SettingsWindow:

Открытые члены

- SettingsWindow (QWidget *parent, [Settings](#) *settings)

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/settingswindow.h
- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/settingswindow.cpp

3.26 Класс StatusFrame

Граф наследования:StatusFrame:

Граф связей класса StatusFrame:

Открытые члены

- StatusFrame (QWidget *parent, Model *model)
- void drawStatus ()

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/statusframe.h
- /home/user/GIT/predator-prey/sources/Predator-prey/GUIApp/statusframe.cpp

3.27 Класс Units

класс для содержания векторов хищников и жертв, а также корма для жертв

```
#include <units.h>
```

Открытые атрибуты

- std::vector< Prey * > preys
preys - вектор указателей на жертву
- std::vector< Predator * > predators
predators - вектор указателей на хищника
- std::vector< Grass * > grass
Units.

3.27.1 Подробное описание

класс для содержания векторов хищников и жертв, а также корма для жертв

Объявления и описания членов классов находятся в файлах:

- /home/user/GIT/predator-prey/sources/Predator-prey/lib/units.h
- /home/user/GIT/predator-prey/sources/Predator-prey/lib/units.cpp