ОТЧЁТ О РАБОТЕ ПРОГРАММЫ

Содержание

1.Задание

Программа «Симулятор жизни уток на озере “Большая охота”»

Долина диких озер прекрасное место для обитания множества разных видов и мастей уток. Все утки очень хвастливы и любят рассказывать все о себе, когда их спросят. Озеро в долине как живой организм знает кто на нем живет, что кто умеет и может об этом рассказать (сколько уток всего, сколько умеет летать/плавать и т. д.). Но иногда в долину приходят охотники, на некоторые дикие озера, которые выбирают случайно и ловят уток, которые там живут один раз в день в течения сезона охоты. Пойманных уток отвозят на домашнее озеро на ферме. Но уткам не нравится ферма, и каждая пойманная утка пытается вернуться на свое родное озеро. Если утка сбежала с фермы и ее опять поймали и привезли на туже самую ферму, то ей подрезают крылья (если летает) или вещают груз на лапку (если плавает), и они больше не могут сбежать. Также в долине каждый день может произойти, что-нибудь необычное… Охотники ловят уток до тех пор, пока на диких озерах не останется ни одной утки, либо не закончится сезон охоты.

ВАРИАНТ #3

В долине озер: 3, уток там живет: 92, а ферм: 1, дней сезона охоты: 9

На Озере 'Киву' живут следующие виды уток:

Утка вид: 'Широконоска', умеет: плавать и знает где она живет, а также имеет атрибуты: имя, вес, форма клюва, здоровье

Утка вид: 'Крохали', умеет: прятаться (охотники не могут ее поймать) и не знает где она живет, а также имеет атрибуты: имя, вес, цвет глаз, окрас крыльев

Утка вид: 'Чернети', умеет: прятаться (охотники не могут ее поймать) и не знает где она живет, а также имеет атрибуты: имя, вес, высота, форма хвоста

На Озере 'Кратерное Озеро' живут следующие виды уток:

Утка вид: 'Шилохвость', умеет: летать и знает где она живет, а также имеет атрибуты: имя, вес, размах крыльев, ширина

Утка вид: 'Желтоклювый чирок', умеет: бездельничать и не знает где она живет, а также имеет атрибуты: имя, вес, размах крыльев, возраст

Утка вид: 'Капский чирок', умеет: мигрировать (если не поймана, перемещается на случайное озеро) и не знает где она живет, а также имеет атрибуты: имя, вес, размер клюва, любимое блюдо

На Озере 'Мьоса' живут следующие виды уток:

Утка вид: 'Хохлатая утка', умеет: летать и знает где она живет, а также имеет атрибуты: имя, вес, здоровье, цвет глаз

Утка вид: 'Крохали', умеет: бездельничать и не знает где она живет, а также имеет атрибуты: имя, вес, окрас хвоста, здоровье

Утка вид: 'Капский чирок', умеет: бездельничать и не знает где она живет, а также имеет атрибуты: имя, вес, выносливость, форма крыльев

Ферма 'COBUMOT' имеет охотников в количестве: 2

Охотник #1 может поймать уток в количестве: 1-6

Охотник #2 может поймать уток в количестве: 3-6

С фермы 'COBUMOT' могут сбежать утки, которые умеют: летать и знают где они живут, но в суматохе попадают на случайное озеро

ДОП. ЗАДАНИЕ:

В долине в случайном месте (озере или ферме) может появиться: НевезучийКряк (Утка, заменить двумя случайными эффекта или героями) на дней: 1

2.Метод решения основной задачи

Для выполнения данной задачи, я посчитал, что буду использовать абстрактный класс, потомков которого потом буду переопределять, классы охотников и озер(озёра реализованы как односвязные списки).

Из библиотек мне понадобились следующие: <iostream>, <string>, <vector>, <ctime> и "windows.h".

2.1.Описание абстрактного класса MamaCryak

Для того, чтобы не описывать каждый класс по отдельности, создадим изначально абстрактный класс MamaCryak, он будет предком для последующих потомков(см. Листинг 2.1).

Листинг 2.1 – абстрактный класс MamaCryak

```
class MamaCryak {
protected:
	string name;
	bool know_home;
	string home;
	int weight;
	MamaCryak* next;
	int fly;
	int swim;
	int migration;
public:
	friend class Ozero;
	friend class Hunter;
	friend void migration(Ozero* a, Ozero* b, Ozero* c);
	MamaCryak() : name(random_name()), know_home(0), home("-"), weight((rand() % 4) + 2), next(NULL),
fly(0), migration(0), swim(0) {};
	string is_name() { return name; };
	void migration_end() { migration = 2; }
	void migration_again() { migration = 1; }
	void off_fly() { fly = 0; }
	void pre_off_fly() { fly = 2; }
	int really_fly() { return fly; }
	void set_name(string a){ name = a; }
	virtual ~MamaCryak() {};
	virtual void get_info() = 0;
	virtual int is_know() = 0;
	virtual int is_fly() = 0;
	virtual int is_swim() = 0;
	virtual int is_hide() = 0;
	virtual int is_chill() = 0;
	virtual int is_migration() = 0;
};
```

Все переменные занесены в protected, дабы можно было наследовать их. У каждой утки есть name(имя), know_home((1)Знает ли утка, где она проживает, если нет, то (0)), home(как раз таки название места обитания),

weight(вес), *next(указатель на следующий объект того-же класса), fly(умеет ли утка летать, (1) если да, (2) если да и уже однажды бежала с фермы и (0) если не умеет), swim(умеет ли плавать) и migration(способна ли утка мигрировать (1) если да, (2) если да, но сегодня уже мигрировала и (0) если нет.

Также объявлены дружественные классы Hunter, Ozero и функция migration, в будущем нам это пригодится(для взаимодействия классов друг с другом). Методы migration_end и migration_again(которые помогают нам изменять данные, находящиеся в protected), off_fly и pre_off_fly тоже изменяют данные в protected, set_name позволяет задать имя, а really_fly и is_name позволяет нам обратиться к protected и вернуть то значение, которое там находится.

Далее идёт определение виртуальных функций, которые мы будем в будущем переопределять для каждого класса, таким образом, объявив их в предке и переопределяя для каждого потомка, мы сможем вызвать эти функции для любого потомка, заранее прописав особенности исполнения этой функции для каждого наследующего данные этого класса классов, это позволит нам учесть все особенности классов-потомков. В моей программе таких потомков 9(с дополнительным заданием 10).

Так же как и у любого класса, у MamaCryak существует конструктор по умолчанию, тут задаются параметры по умолчанию, например вес, который выбирается случайно от 2-5 кг, указатель на следующую утку, который по умолчанию NULL, home(дом) у уток по умолчанию отсутствует, но на всякий случай проинициализируем его как “-“, а такие параметры как migration(способность мигрировать), swim(способность плавать) и know_home(знает ли где живёт), их по умолчанию ставим в 0, а потом, если нам понадобится, сможем изменить на 1(1-умеет, 0-не умеет).Чуть-чуть подробнее хочется сказать о присваивании имени каждой утке. Оно задается функцией random_name,давайте рассмотрим её поподробнее(см Листинг 2.2).

Листинг 2.2 – функция генерации имени random_name

```
string random_name() {
	int a = rand() % random;
	string bckp;
	bckp = names[a];
	names.erase(names.begin() + a);
	random -= 1;
	return bckp;
}
```

Эта функция выбирает из динамического массива вектор(заранее заполненного 100 случайных имён для уток) случайное имя, затем удаляя его из массива, таким образом, у всех уток разные имена, каждая из них, особенная.

В итоге у нас есть готовый абстрактный класс, который будет использоваться для каждого потомка, описывая их общие характеристики.

2.2.Описание потомков абстрактного класса

Всего в игре 3 озера, на каждом озере по 3 типа уток, следовательно всего типов 9(10 с дополнительным заданием, но об этом потом). В задании были одинаковые названия вида у уток, потому я поменял некоторые из них, добавив приставку бета(Крохали бета и Капский чирок бета)

2.2.1. Добавление полей потомкам

Так как каждый из потомков обладает своими особенными атрибутами(форма клюва(kluvs), здоровье(health), цвет глаз(color_eye), окрас крыльев(color_wings), высота(height), форма хвоста(tail_form), размах крыльев(razmah), width(ширина), age(возраст), chill(умение бездельничать), kluv_size(размер клюва), bludo(любимое блюдо), resistance(выносливость), wing_form(форма крыльев) и tail_color(цвет хвоста).

Большинство, но не все из этих атрибутов требуется добавить для каждого потомка, в соответствии с видом утки(например умение бездельничать(chill) держать как переменную не имеет смысла, т.к мы можем просто переопределить функцию, об этом будет далее). Тип переменных

будет зависеть от того, что из себя представляет характеристика(например форма клюва это определенно какое-нибудь строковое значение(string), а вот например возраст уже целочисленное значение(к примеру int)).

Так же, все эти переменные должны каким то образом принимать случайные значения. В случае с числовыми значениями мы будем использовать rand() (впрочем, как и с весом утки у абстрактного класса), а вот с строковыми значениями поступим так, заведем массивы string, в которые заранее пропишем возможные варианты). Как пример приведу массив любимых блюд(см. Листинг 2.3).

Листинг 2.3 – пример массива для присваивания потомкам случайного любимого блюда (значения типа string)

```
string bludos[10] = { "раки", "моллюски", "червяки", "личинки комаров", "горох", "пшеница", "ячмень", "рожь", "бобы", "творог" };
```

Таким образом мы описываем конструктор для каждого типа утки, в соответствии с её особенностями, инициализируя поля случайными значениями, так каждая утка становится немножечко особенной.

2.2.2.Переопределение виртуальных функций, таких как is_fly, is_swim, is_hide и т.д.

В родительском абстрактном классе MamaCryak обозначили виртуальные функции, настало время их переопределять. Но для того, чтобы переопределять, нужно понять, для чего они нам нужны. Эти функции будут возвращать нам 1 если утка умеет это, и 0 если не умеет. Для чего нам это нужно? Для того, чтобы мы смогли посчитать и показать в общей характеристике озёр количество уток, умеющих что-то, или же, чтобы определить, умеет ли утка делать это. Например, в методе охотника Hunt нам потребуется обратиться к утке и узнать умеет ли она прятаться, от этого будет зависеть ход охоты, поймают утку или же нет).

2.2.3.Переопределение виртуальной функции get_info

Давайте рассмотрим на примере 1ого из классов, например на классе Shilohvost. Утки этого вида по заданию могут летать, знают где они живут, имеет имя, вес, размах крыльев и ширину. Заострим внимание на функции get_info, это как раз таки та функция, которая определяет, как рассказывает о себе каждая утка. У каждой утки эта функция особенная, ведь каждая утка обладает своими атрибутами. У нашей утки например, как сказано выше, это размах крыльев и ширина, именно потому это утка, рассказывая о себе, говорит об этом(см. Листинг 2.4).

Листинг 2.4 – пример потомка Shilohvost

```
class Shilohvost : public MamaCryak {
        int razmah;
        int width;
public:
        Shilohvost() : razmah(50 + (rand() % 41)), width(15 + (rand() % 21)) {
                know_home = 1;
                fly = 1;
        };
        virtual ~Shilohvost() {};
        virtual int is_fly() {
                if ((this->fly == 1) || (this->fly == 2)) {return 1;}
                else {return 0;}
        }
        virtual int is_swim() {return 0;}
        virtual int is_hide() {return 0;}
        virtual int is_chill() {return 0;}
        virtual int is_migration() {return 0;}
        virtual int is_know() {return 1;}
        virtual  void get_info() {
                cout << "Приветики, я  " << name << ", вида Шелохвость." << endl;
                cout << "Что то я поправилась, вешу аж " << weight << " кило! " << endl;
                if (fly == 1) {cout << "Хорошо что я умею летать" << endl;}
                else  if (fly == 2) {cout << "Однажды охотники поймали меня, но так как я умею летать, я
смогла бежать." << endl;}
                else {cout << "Охотники дважды ловили меня, после чего подрезали мне крылья, теперь я не
умею летьать." << endl;}
                cout << "Господи, и в ширину я уже " << width << " см." << endl;
                cout << "Зато размах крыльев " << razmah << " см." << endl;
                cout << "Хорошо что хоть живу на " << home << endl;
                cout << "Мне пора!" << endl;
                cout << endl;
        }
};
```

Таким образом, каждая утка способна рассказать все о себе, учитывая все свои особенности, показать нам, что она особенная.

2.3.Описание класса Ozero

Где у нас обитают утки? –На озёрах. Каждое озеро будет представлять из себя список, в котором собственно и будут находиться наши утки. Каждый объект типа Ozero имеет указатель на начало списка(*head), название самого озера(или же фермы)(name_ozero) и переменные типа int, которые показывают различные параметры озера, такие как: количество уток на озере, количество уток умеющих плавать, летающих, умеющих прятаться, умеющих бездельничать, умеющих мигрировать и знающих где они живут уток(size_list, count_swim, count_fly, count_hide, count_chill, count_migration, int count_is_know). Благодаря этим переменным мы сможем выдать точную иныормацию по каждому из озер/ферм.

Так же в классе Ozero прописан конструктор по умолчанию. Чтобы создать объект класса Ozero нужно при создании задать ему имя. Все переменные по умолчанию принимают значение 0(в случае с указателем NULL).

При работе с динамической памятью очень важно в конце программы очищать используемую память, потому, помимо вышеперечисленного, в классе Ozero присутствует и деструктор. Он вызывается в конце работы программы и освобождает память, использованную для создания списка, который содержит в себе уток(см. Листинг 2.5).

Листинг 2.5 – деструктор класса Ozero

```
~Ozero() {
		MamaCryak* tmp = NULL;
		if (head != NULL) {
			while (head->next != 0) {
				tmp = head;
				head = head->next;
				delete tmp;
				tmp = NULL;
			}
			delete head;
		}
	}
```

Теперь же можем перейти к описанию основных методов для класса Ozero.

2.3.1.Методы

Что нам нужно делать с утками, чтобы выполнить поставленное задание? Мы должны уметь забирать утку из озера/фермы, добавлять её на какое-нибудь другое озеро/ферму и обращаться к утке, дабы проверить, что она умеет и какими характеристиками обладает и выводить информацию по озеру. Начнем с метода добавления утки в список.

Метод add_duck может нам добавить утку в определённое озеро или же ферму. Аргументом функции будет указатель на абстрактный класс. Добавлять мы будем в конец списка. Работает он очень просто. Если список пуст(head=NULL), то указатель head приравниваем к передаваемому аргументу в нашу функцию, а именно к node(*MamaCryak), после чего присваиваем утке по *node значение поля *next = NULL. После чего, используя виртуальные функции изменяем значения умеющих или обладающих какими-нибудь характеристиками уток на озере. А если же в списке уже есть какие либо утки, то мы заходим в цикл while, который переходит к следующей утке до того момента, пока указатель на следующую утку не будет = NULL(на всякий случай проверяем и ненулевой ли указатель на саму утку). Затем присваиваем последней утке в списке указатель на добавляемую утку, тем самым добавив нашу утку в конец списка. Далее так же изменяем значения уток с способностями и особенностями на этом озере.

Метод del_duck позволяет нам удалять утку по её номеру в списке. Аргументом функции является номер удаляемой утки. Как она работает – для начала мы создаем 2 указателя на MamaCryak tmp и prev. Для начала приравниваем значение tmp к head(теперь tmp указывает на начало нашего списка). Далее проверяем, какой элемент нам нужно удалить, если это 0вой элемент(первый), то указателю head присваиваем значение следующего head(т.е. указатель head теперь указывает на ту утку, которая была второй в списке, т.к. удалив первую утку она станет первой). Затем обнуляем указатель на следующий элемент для нашей удаляемой утки(под указателем

tmp). Или входим в цикл while, который работает до того момента, пока мы не дойдем по номеру до утки, которая находится перед той, которую хотим удалить. Когда доходим, то приравниваем prev к tmp, tmp придаем указатель на следующую утку, затем для утки под указателем prev(той что стояа до удаляемой) присваиваем значению указателя next указатель на tmp->next(на утку которая была сразу после удаляемой). Далее опять же обнуляем указатель на следующую утку для удаляемой утки (присваиваем NULL для *next) Ну и раз утка удалена, то должна и измениться статистика по озеру, потому вычитаем из наших значений то, что умела делать утка, благодаря описанным ранее виртуальным функциям. Важно отметить, что функция имеет тип возвращаемого значения, а именно на указатель абстрактного класса MamaCryak. Таким образом удаляя утку. мы на самом деле не удаляем её, а лишь вытаскиваем из списка.

Метод choose_duck позволяет нам вернуть объект так называемой утки, находящейся на определённом озере и под определённым индексом. В нашей программе нам нужно будет проверять некоторые свойства утки под определённым номером и, дабы не удалять утку из озера а потом добавлять обратно, мы прописываем этот метод. Работает он по аналогии с del_duck, только ничего не меняет, просто возвращает нам нашу утку под определённым номером. Таким образом, например, мы можем проверить, умеет ли выбранная утка летать, прятаться и т.д.

Но также, кроме вышеперечисленного, нам нужен метод вывода информации по каждому озеру/ферме. get_information как раз-таки и отвечает за это, он просто выводит нам численные значения количества уток с той или иной способностью или характеристикой.

Так же есть небольшой метод cut_wings, который уменьшает количество умеющих летать уток на ферме(он нам нужен т.к. счётчики обновляются при добавлении в список, а крылья утке подрезают уже на ферме).

Описав 3 вышеперечисленные функции, мы получаем всё, что требуется для нашей программы, мы можем реализовать побег с фермы, охоту, или, например, миграцию. Кроме как добавления в список, удаления и выбора по индексу нам больше ничего не потребуется(можно было не прописывать функцию выбора, но тогда, например при охотке мы бы сначала удаляли утку из озера, проверяли может ли она спрятаться и удастся ли это ей, и если удастся добавляли обратно. Согласитесь, не совсем разумно). Более подробно о вышеперечисленных методах(см. Приложение 1).

2.4.Описание класса Hunter

Так же, помимо уток и озер/ферм у нас есть и охотники. Каждый охотник имеет своё имя, свою ферму, максимально возможное число пойманных уток и минимально возможное число пойманных уток. Сам класс без метода Hunt выглядит так(см.Листинг 2.6).

Листинг 2.6 – класс Hunt

```
class Hunter {
	int max_frags;
	int min_frags;
	string farm;
	string name;
public:
	Hunter(int max_frags, int min_frags, string farm, string name) : max_frags(0), min_frags(0), farm("~"), name("~") {
		this->max_frags = max_frags;
		this->min_frags = min_frags;
		this->farm = farm;
		this->name = name;
	};
```

Как видно из листинга, этот класс обладает 2 численными переменными(максимальное и минимальное число пойманных уток) и 2 строковыми(ферма и имя).Теперь можем перейти к описанию одного из основных методов в нашей программе.

2.4.1.Описание метода Hunt

Охота проходит таким образом, сначала определяется сколько уток охотник планирует поймать, после высвечивается сообщение о том какую утку собирается поймать охотник, затем проверяется, умеет ли утка прятаться, если да, то с шансом 75% она сможет спрятаться, в ином исходе

охотник ловит её и отправляет жить на ферму(программа выдаст сообщение). Это всё повторяется до тех пор, пока охотник не попытается поймать столько уток, сколько был намерен в начале охоты. После выводится число с количеством пойманных уток, на этом охота для охотника заканчивается. Если охотник собирался поймать больше уток чем есть на озере(там точно кто-то есть, на пустое озеро он не пойдёт) то он попробует словить столько уток, сколько есть на озере(см. Приложение 2).

2.5.Функция migration

Функция миграция позволяет уткам, умеющим мигрировать непосредственно мигрировать. В консоль выводится сообщение о начале миграции. Мы проходимся по 3 озёрам, определяем есть ли у утки способность мигрировать, шанс что она попадет на каждое озеро 33.33%(т.е. есть шанс в 33.33% что утка останется на озере и 66.67% что она мигрирует). После того, как утка мигрирует, она больше не может мигрировать в эту стадию миграции(это сделано для того, чтобы утка при миграции с 1 ого озеро на 2ое, при прохождении по 2ому озеру не мигрировала вновь). В конце дня возвращаем способность мигрировать уткам которые это умели, на этом стадия миграции заканчивается(см. Приложение 3).

2.6.Функция pobeg

Так же у некоторых уток есть возможность бежать с фермы. Функция побега как раз таки и реализует это. Сначала проверяется. Соответствует ли утка тем условиям, которые требуются для побега, затем проверяется, была ли поймана утка ранее, если да, то ей подрезают крылья, и больше она не сбежит. Если утка соответствует всем вышеперечисленным условие то с 50% шансом она сбегает. Когда утка сбегает, она получает fly = 2(говорит о том, что если её поймают в следующий раз, то ей подрежут крылья и она больше не сбежит), а затем она выбирает на какое озеро из 3 мигрировать, шанс одинаковый(см. Приложение 4). Теперь, когда у нас реализованы все требуемые классы и функции перейдём к нашей функции int_main.

2.7.Описание функции int_main

В начале нашёл функции мы создаем наши озёра и фермы, охотников и различные вспомогательные переменные. Далее заполняем озёра случайными типами уток(количество уток на озёрах определяется заранее). После чего начинается наш цикл, который проходит 9 раз, как раз таки столько дней охоты у нас.

В начале цикла описано как раз таки наше так называемое меню(см. приложение 5). Работает оно так: нам предлагается выбрать 1 из 3 пунктов.

Выбрав 1 пункт(введя 1) мы можем посмотреть информацию по конкретной утке на конкретном озере. После выбора 1 нам предлагается выбрать озеро на котором мы хотим посмотреть уток. После выбора озера нам показывается диапазон уток, к которому мы можем обратиться(введя номер этой самой утки), тогда она расскажет нам о себе. После чего при вводе 1 мы возвращаемся в главное меню.

Выбрав 2 пункт(введя 2) мы можем посмотреть информацию по конкретному озеру или ферме. После выбора 2 нам прилагается выбрать озеро, по которому мы хотим посмотреть статистику. Выбирая озеро нам выводится статистика по этому озеру. После чего при вводе 1 мы возвращаемся в главное меню.

Выбрав 3 пункт(введя 0) мы начинаем день охоты.

Меню будет выводится до тех пор, пока мы не начнём день охоты.

День охоты проходит так. Сначала охотники отправляются на охоту. Сначала 1 охотник выбирает на какое озеро пойдёт, после чего охотится. Затем 2ой охотник делает тоже самое. После проверяется, остались ли утки на свободе, если не осталось, то программа завершается и охотники побеждают, если нет, то двигаемся дальше.

После начинается стадия миграции.

Затем следует стадия побега.

Всё это продолжается на протяжении 9 дней после чего, если программы не завершилась преждевременно победой охотников, то побеждают утки, после чего на экран выводится краткая статистика.

На этом наша игра окончена!(см. Листинг 2.7, или 2.8 при победе охотников)

Листинг 2.7 – победа уток

```
cout << "~~~~~~~~~~~~~~~~~~~~" << endl;
	cout << "| Утки победили!!! |" << endl;
	cout << "~~~~~~~~~~~~~~~~~~~~" << endl;
	cout << "В сумме на озерах осталось " << kivu.get_size()+
kraternoe_ozero.get_size()+ mjosa.get_size() << " уток" << endl;
	cout << "А именно:" << endl;
	cout << "На озере Киву: " << kivu.get_size() << " уток" <<endl;
	cout << "На Кратерном озере: " << kraternoe_ozero.get_size() << " уток" << endl;
	cout << "На озере Мьоса: " << mjosa.get_size() << " уток" << endl;
	cout << "А вот на ферме COBUMOT осталось: " << COBUMOT.get_size() << " уток" <<
endl;
		return 0;
```

Листинг 2.8 – победа охотников

```
cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
cout << "| Охотники победили, все утки пойманы! |" << endl;
cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
```

3.Метод решения дополнительного задания

По заданию нам нужно придумать НевезучемуКряку 2 особые способности. У меня они таковы:

1 – на то озере с Кряком, умирает половина уток(см Листинг 3.2).

2 – в этот день минимальное число пойманных уток у охотников увеличивается в 3 раза, а максимальное в 2(см Листинг 3.1).

Этот кряк может появиться с 50% шансом начиная с 1 дня. Появиться он может только 1 раз за всю игру. Он появляется вначале дня, и умирает в начале следующего. Как только он появляется на определённом озере , утки сразу начинают умирать. Для того, чтобы временно увеличить показатели охотников я добавил в класс охотников новые методы, которые как раз и изменяют максимальное и минимальное число пойманных уток, а в последствии восстанавливают изначальные значения.

Листинг 3.1 – методы для охотников

```
void boost_hunter() {
		min_frags = min_frags * 3;
		max_frags = max_frags * 2;
	}
void boost_reset() {
		min_frags = min_frags / 3;
		max_frags = max_frags / 2;
	}
```

Листинг 3.2 – фукция убийства уток

```
void kill_duck(Ozero *a) {
	MamaCryak* duck;
	int count = 0, c, b = a->get_size();
	b = b / 2;
	for (int i = 0; i < b; i++) {
		c = rand() % b + 1;
		duck = a->delete_duck(c);
		c--;
		cout << "Как жаль, но сегодня скончаласть утка " << duck->is_name() << "
почему же ей так не повезло....." << endl;
		Sleep(1000);
		delete duck;
		count++;
	}
	cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
<< endl;
	cout << "Вот это неудача, просто  скончалось " << count  << " обычных уток, такое
невезение точно неспроста....." << endl;
	cout << "(Всего в игре осталось " << 93 - count  << " обычных уток)" << endl;
	Sleep(5000);
	system("cls");
};
```

4.Пример работы программы

При запуске симулятора жизни уток выводится баннер с названием(см. Рис. 4.1).

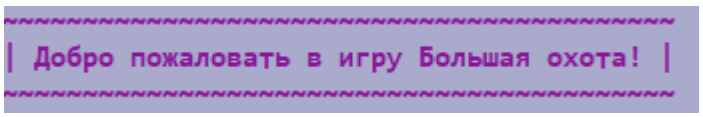

Рис. 4.1 - Начало программы

Далее попадаем в главное меню (см. Рис. 4.2).

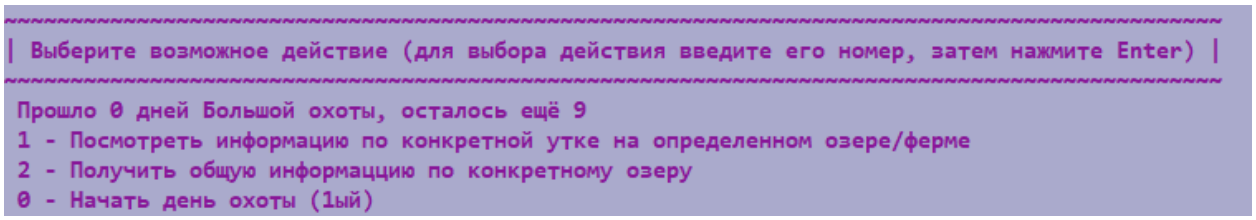

Рис 4.2 – Главное меню

Введя 1 можем выбрать 1 из озер(см. Рис. 4.3).

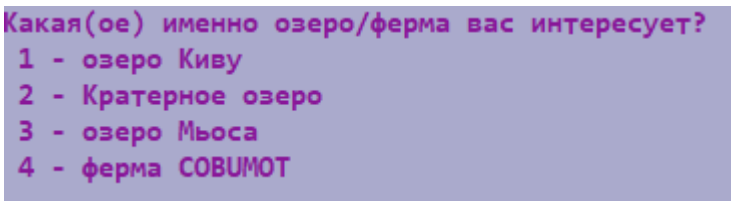

Рис 4.3 – Выбор озера при желании обратиться к утке

И введя номер утки обращаемся к ней(см. Рис. 4.4).

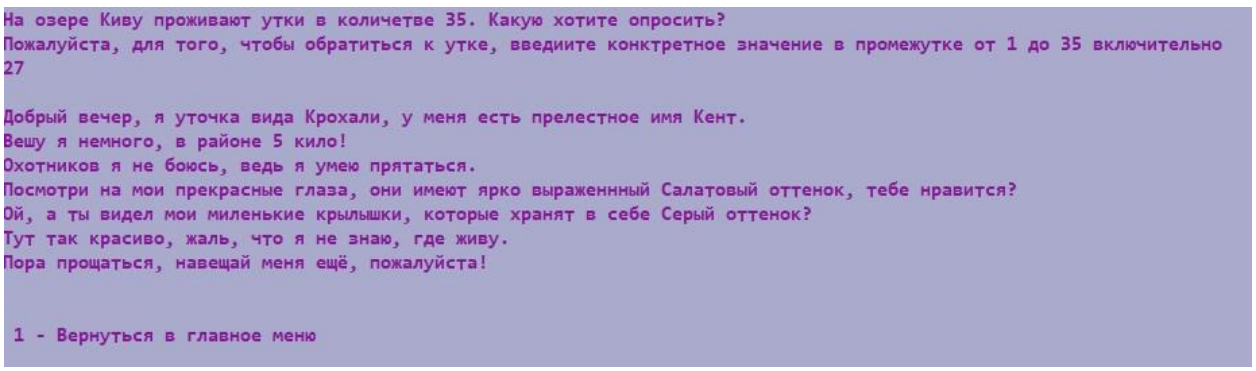

Рис 4.4 – Обращаемся к утке по индексу

Возвращаемся в главное меню, вводим 2, выбираем 2 озеро, опять же введя 2, и получаем статистику по Кратерному озеру(см. Рис. 4.5).

Рис 4.5 – Просматриваем общую статистику по озеру

Начнём же день охоты и посмотрим, как он проходит:

1ый охотник выходит на охоту(см. Рис. 4.6);

2ой охотник выходит на охоту(см. Рис. 4.7);

Стадия миграции(см. Рис. 4.8);

Стадия побега(см. Рис. 4.9).

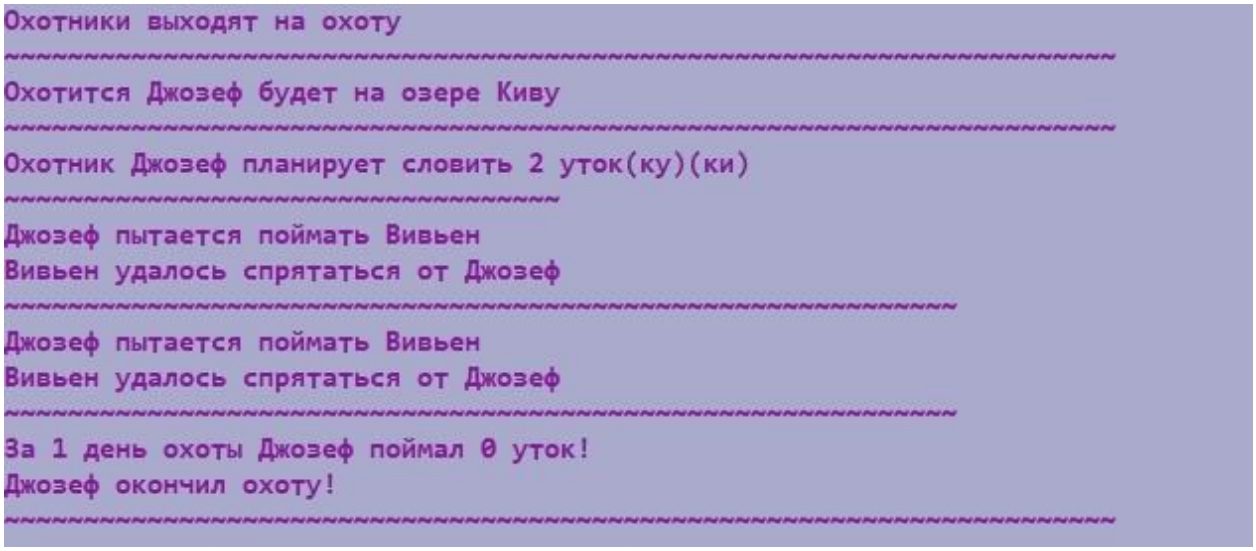

Рис 4.6 – Первый охотник выходит на охоту

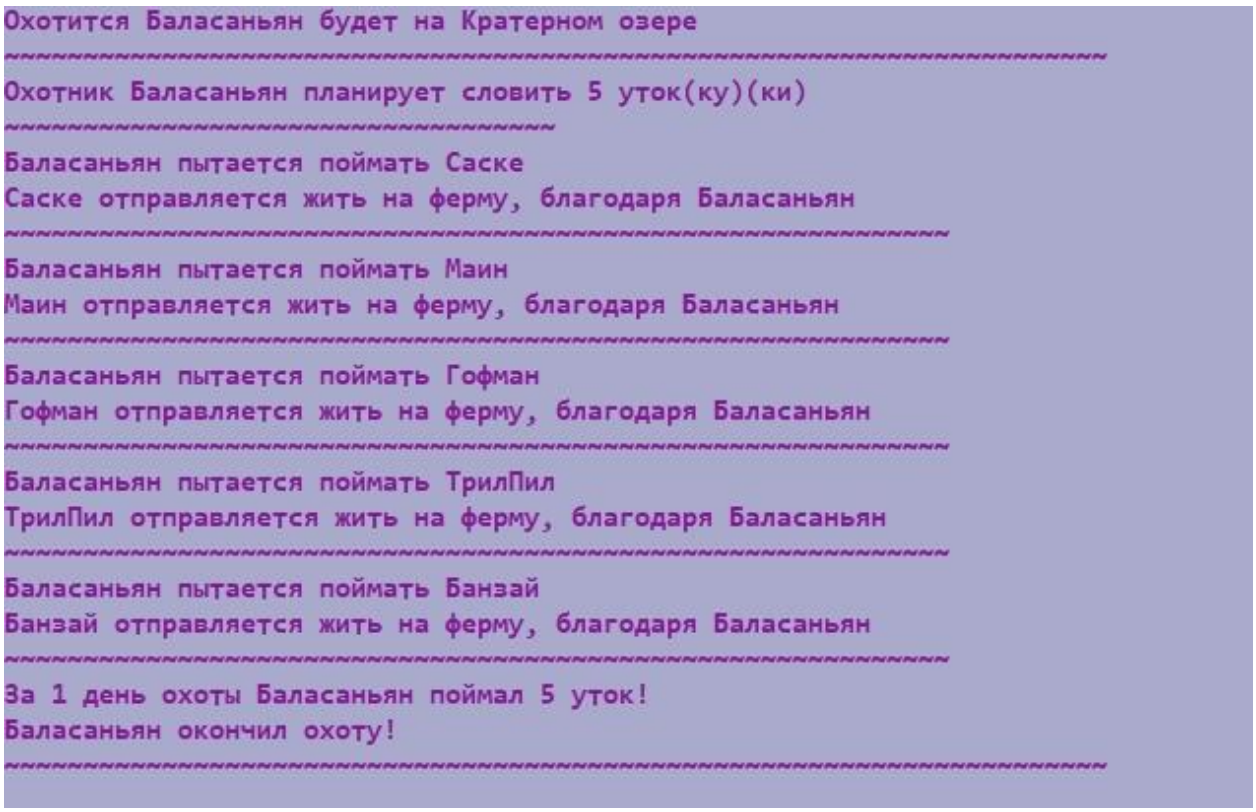

Рис 4.7 – Второй охотник выходит на охоту

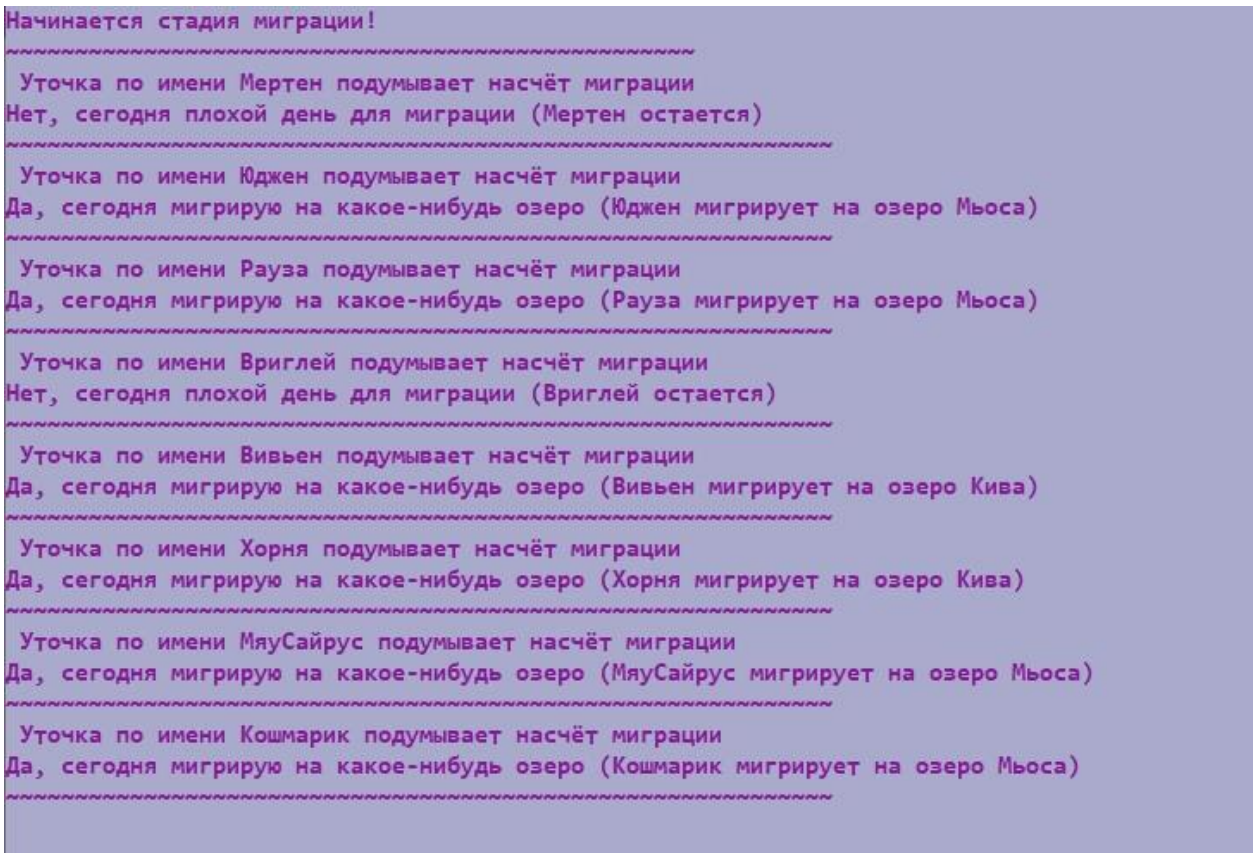

Рис 4.8 – стадия миграции

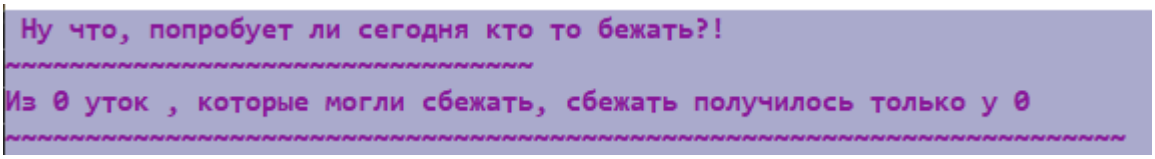

Рис 4.9 – стадия побега

Вот и закончен день охоты, проверим что там на ферме(см. Рис. 4.10)

Рис 4.10 – ферма после 1 дня охоты

Начнём же 2ой день

Появление кряка и убийство половины озера(см. Рис. 4.11).

1ый охотник выходит на охоту(см. Рис. 4.12);

2ой охотник выходит на охоту(см. Рис. 4.13);

Стадия побега(см. Рис. 4.14).

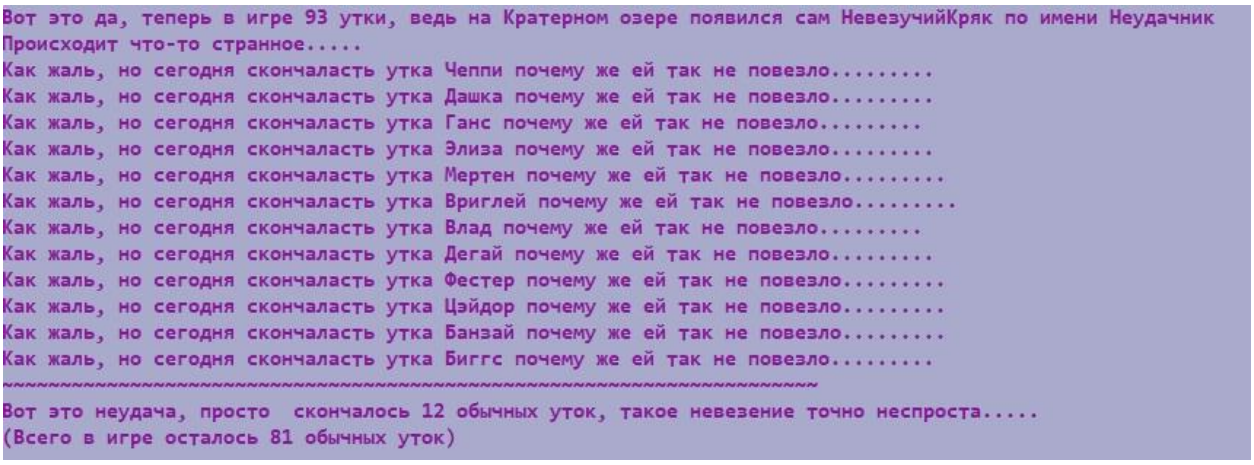

Рис 4.11 - Появление кряка и убийство половины озера

Рис 4.12 – Первый охотник выходит на охоту

Рис 4.13 – Второй охотник выходит на охоту

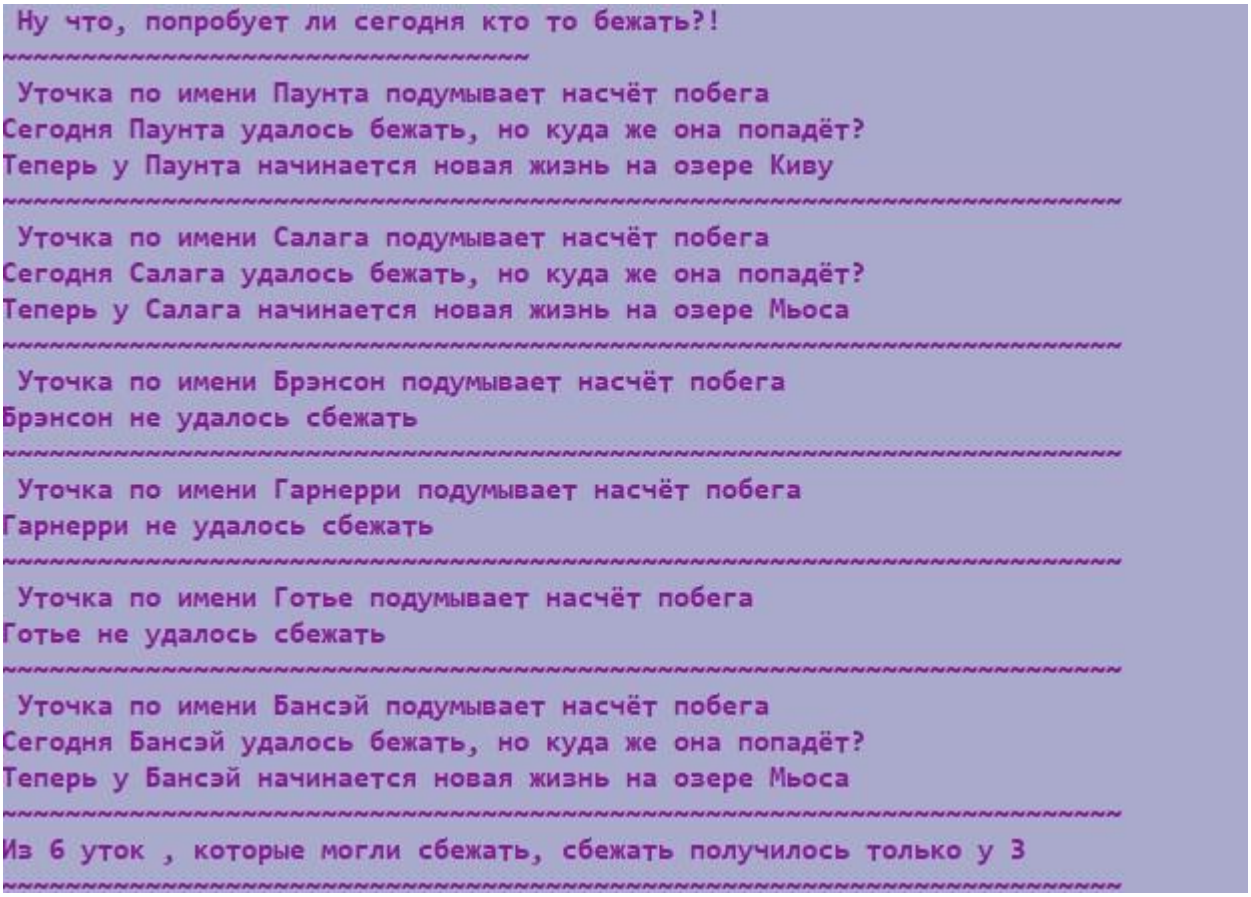

Рис 4.14 – Стадия побега

Запустим же 3ий день и проверим, может произойдет что то интересное. В начале дня НевезучийКряк покидает нас(навсегда)(см. Рис. 4.15), затем как мы можем видеть показатели охотников вернулись в норму, после буста от НевезучегоКряка)(см. Рис. 4.16 и Рис. 4.17, тут кстати поймали всех уток на 2 озере) и стадия побега, где утку поймали 2жды и подрезали ей крылья(см. Рис. 4.18)

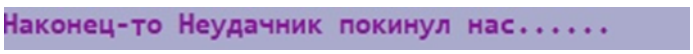

Рис 4.15 – Смерть НевезучегоКряка

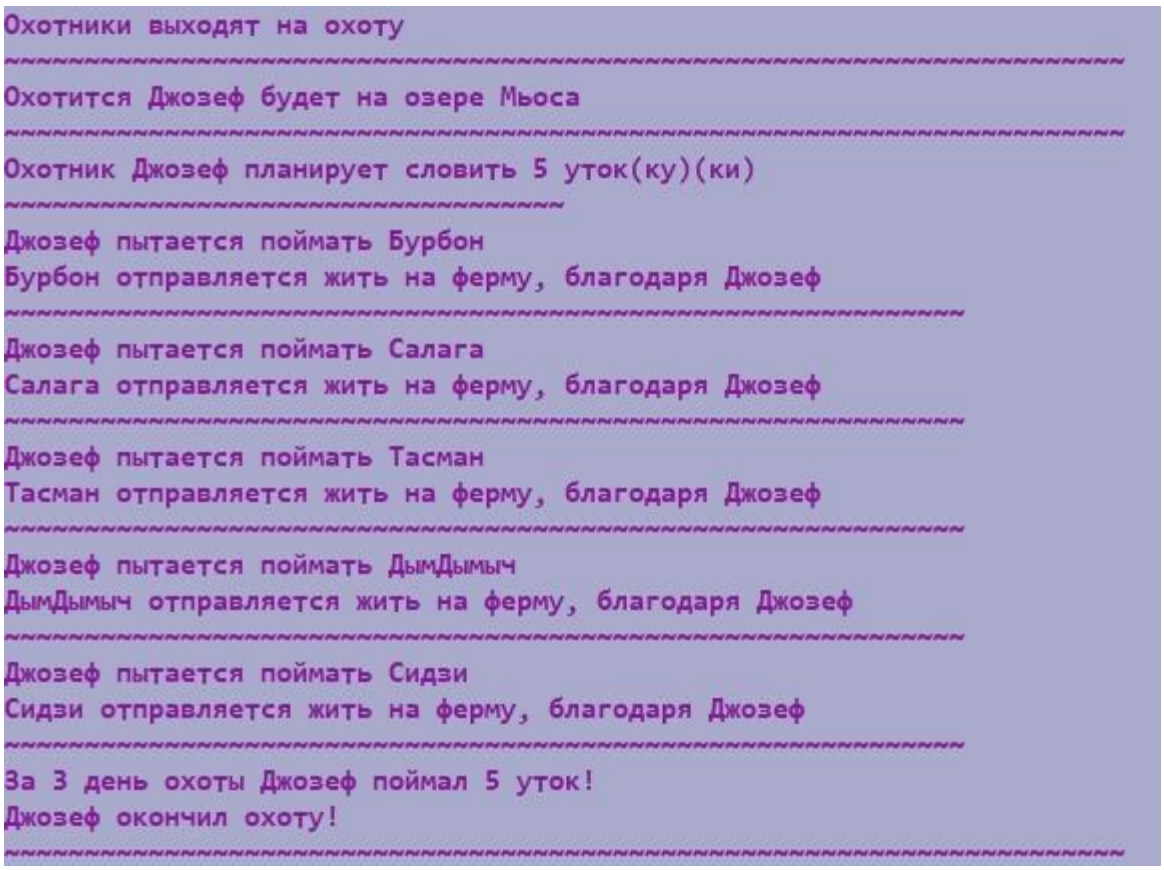

Рис 4.16 – Первый охотник выходит на охоту

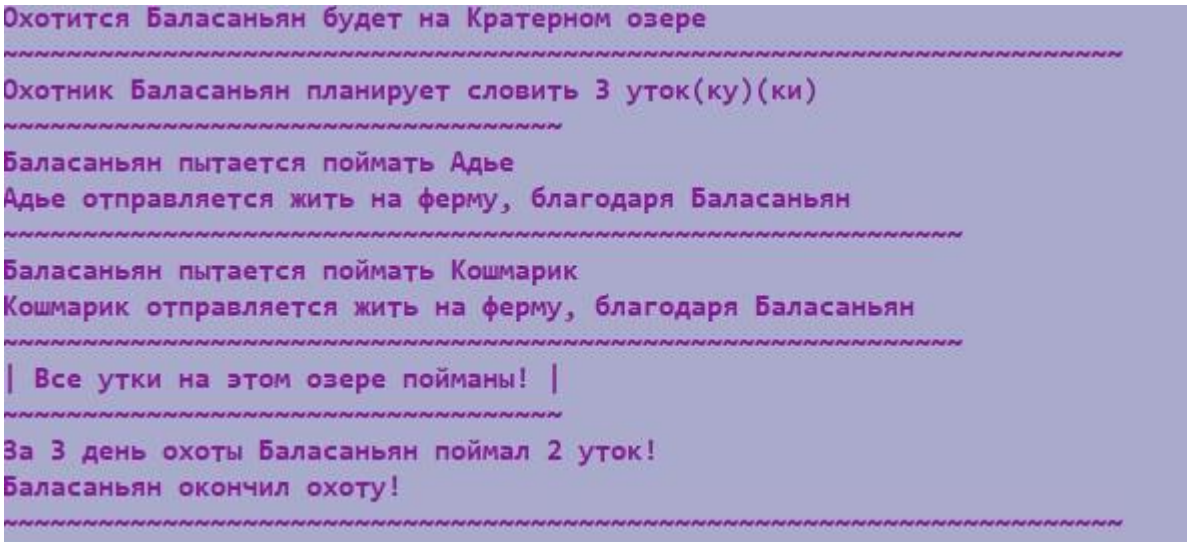

Рис 4.17 – Второй охотник выходит на охоту

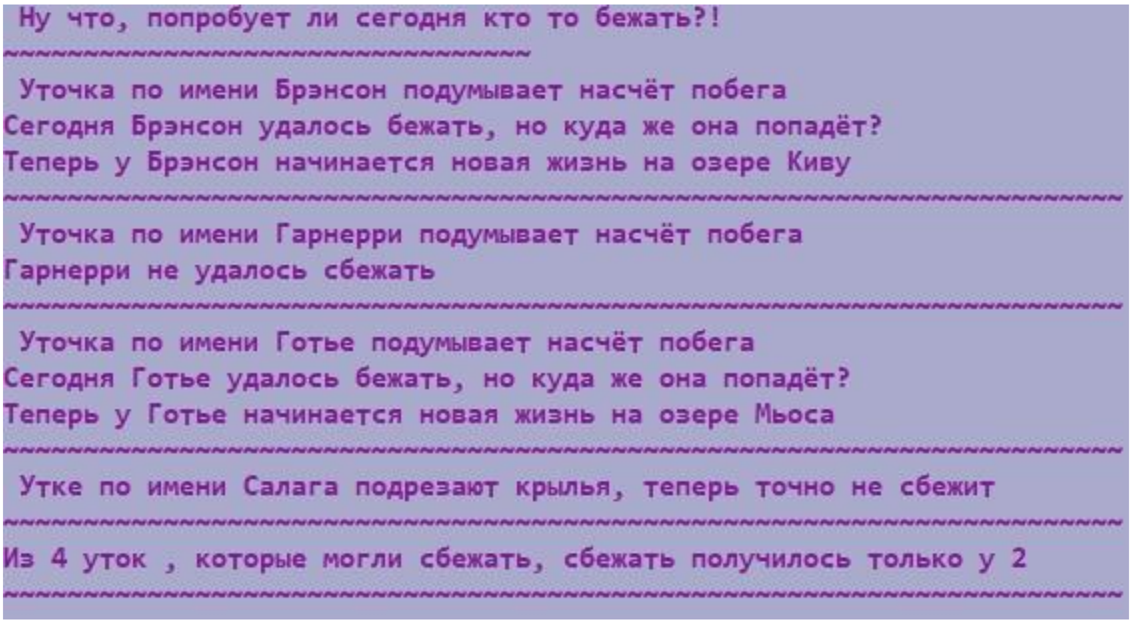

Рис 4.18 – Стадия побега

Проверяем, действительно ли на Кратерном озере осталось 0 уток(см. Рис. 4.19)

Рис 4.19 – Статистика 2ого озера на 3ий день охоты

Посмотрим, чем кончится наша игра!(см. Рис 4.20)

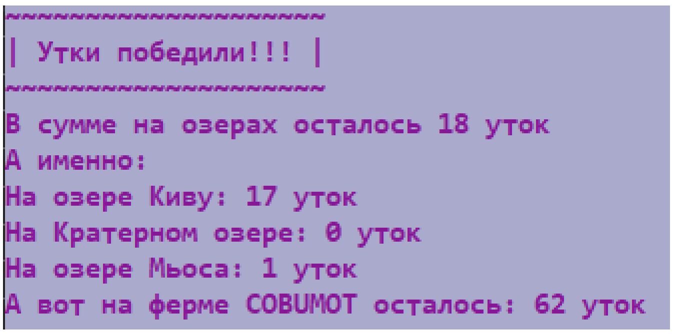

Рис 4.20 – В этот раз победу одержали утки!

5.Среда разработки

Microsoft Visual Studio 2019

Сам код находится в Приложении 6.

Список использованных источников

1. Шилдт, Герберт C++ для начинающих. Шаг за шагом / Герберт Шилдт. - М.: ЭКОМ Паблишерз, 2019. – с. 640.

2. Прата, Стивен Язык программирования C++. Лекции и упражнения / Стивен Прата. - М.: Вильямс, 2014. – с. 213.

3. Шилдт, Герберт C++: базовый курс / Герберт Шилдт. - М.: Вильямс, 2008. – с. 624

4. Страуструп, Программирование. Принципы и практика использования C++ / Бьерне Страуструп. - М.: Вильямс, 2018. - с. 1328.

5. Седжвик, Алгоритмы на C++. Анализ структуры данных. Сортировка. Поиск. Алгоритмы на графах / Роберт Седжвик. - М.: Вильямс, 2019. - с. 1056.

6. Скотт, Эффективный и современный C++: 42 рекомендации по использованию C++11 и C++14 / Мейерс Скотт. - М.: Вильямс, 2019. - с. 304.

Приложение 1

```
1.void cut_wings() {
2.    count_fly--;
3.}
4.void add_duck(MamaCryak* node) {
5.    MamaCryak* tmp = NULL;
6.    if (head == NULL) {
7.        head = node;
8.        head->next = NULL;
9.        size_list++;
10.             count_swim += (node->is_swim());
11.             count_fly += (node->is_fly());
12.             count_hide += (node->is_hide());
13.             count_chill += (node->is_chill());
14.             count_migration += (node->is_migration());
15.             count_is_know += (node->is_know());
16.         }
```

```
17.         else {
18.             tmp = head;
19.             while ((tmp) && (tmp->next)) {
20.                 tmp = tmp->next;
21.             }
22.             if ((tmp != NULL) && (tmp->next == NULL)) {
23.                 tmp->next = node;
24.                 size_list++;
25.                 count_swim += (node->is_swim());
26.                 count_fly += (node->is_fly());
27.                 count_hide += (node->is_hide());
28.                 count_chill += (node->is_chill());
29.                 count_migration += (node->is_migration());
30.                 count_is_know += (node->is_know());
31.             }
32.         }
33.         if (node->know_home == 1) {
34.             node->home = name_ozero;
35.         }
36.     }
37.     int get_size() {
38.         return size_list;
39.     }
40.     void get_information() {
41.         cout << "В общей сложности проживает " << size_list << " уток"
    << endl;
42.         cout << "Из них: " << endl;
43.         cout << count_swim << " умеет(ют) плавать," << endl;
44.         cout << count_fly << " умеет(ют) летать," << endl;
45.         cout << count_hide << " умеет(ют) прятаться," << endl;
46.         cout << count_chill << " умеет(ют) бездельничать," << endl;
47.         cout << count_migration << " умеет(ют) мигрировать," << endl;
48.         cout << count_is_know << " знает(ют) где живет(ут)" << endl;
49.     }
50.     MamaCryak* delete_duck(int i) {
51.         i--;
52.         MamaCryak* prev, * tmp;
53.         tmp = head;
54.         if (i == 0) {
```

```
            head = head->next;
            tmp->next = NULL;
            size_list--;
            count_swim -= (tmp->is_swim());
            count_fly -= (tmp->is_fly());
            count_hide -= (tmp->is_hide());
            count_chill -= (tmp->is_chill());
            count_migration -= (tmp->is_migration());
            count_is_know -= (tmp->is_know());
            return tmp;
        }
        while (i != 1) {
            tmp = tmp->next;
            i--;
        }
        prev = tmp;
        tmp = tmp->next;
        prev->next = tmp->next;
        tmp->next = NULL;
        size_list--;
        count_swim -= (tmp->is_swim());
        count_fly -= (tmp->is_fly());
        count_hide -= (tmp->is_hide());
        count_chill -= (tmp->is_chill());
        count_migration -= (tmp->is_migration());
        count_is_know -= (tmp->is_know());
        return tmp;
    };
    MamaCryak* choose_duck(int i) {
        i--;
        MamaCryak* tmp;
        tmp = head;
        if (i == 0) {
            return tmp;
        }
        while (i != 0) {
            tmp = tmp->next;
            i--;
        }
```

```
94.         return tmp;
95.     };
```

Приложение 2

```
1.int Hunt(Ozero* ozero, Ozero* farm) {
2.         MamaCryak* b;
3.         int kills, dead = 0;
4.         int a, m, l;
5.         a = ozero->get_size();
6.         if (a == 0) {
7.             cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ " << endl;
8.             cout << "| Все утки на этом озере пойманы! |" << endl;
9.             cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
10.                 Sleep(2000);
11.                 return dead;
12.             }
13.             m = max_frags - min_frags;
14.             l = 1 + (rand() % a);
15.             kills = min_frags + (rand() % (m));
16.             cout << "Охотник " << name << " планирует словить " <<
    kills << " уток(ку)(ки)" << endl;
17.             cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
18.             Sleep(1000);
19.             for (int i = 0; i < kills; i++) {
20.                 if (ozero->get_size() == 0) {
21.                     cout << "| Все утки на этом озере пойманы! |" <<
    endl;
22.                     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" <<
    endl;
23.                     return dead;
24.                     Sleep(2000);
25.                 }
26.                 l = (1 + rand() % a);
27.                 b = ozero->choose_duck(l);
28.                 cout << name << " пытается поймать " << b->name <<
    endl;
29.                 Sleep(1000);
```

```
30.                 if ((rand() % 4 != 0) && (b->is_hide() == 1)) {
31.                     cout << b->name << " удалось спрятаться от " <<
   name << endl;
32.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ " <<
   endl;
33.                     Sleep(1000);
34.                 }
35.                 else {
36.                     b = ozero->delete_duck(l);
37.                     cout << b->name << " отправляется жить на ферму,
   благодаря " << name << endl;
38.                     farm->add_duck(b);
39.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ " <<
   endl;
40.                     Sleep(1000);
41.                     dead++;
42.                 }
43.                 a = ozero->get_size();
44.             }
45.             return dead;
46.         }
```

Приложение 3

```
1.void migration(Ozero* a, Ozero* b, Ozero* c) {
2.     MamaCryak* duck;
3.     cout << "Начинается стадия миграции!" << endl;
4.     cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" <<
   endl;
5.     Sleep(2000);
6.     for (int i = 1; i <= a->get_size(); i++) {
7.         duck = a->choose_duck(i);
8.         if ((duck) && (duck->is_migration() == 1)) {
9.             cout << " Уточка по имени " << duck->is_name() << "
   подумывает насчёт миграции" << endl;
10.                 Sleep(1000);
```

```
                if (rand() % 3 == 0) {
                    cout << "Нет, сегодня плохой день для миграции (" << duck->is_name() << " остается)" << endl;
                    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
                    duck->migration_end();
                    Sleep(1000);
                }
                else if (rand() % 3 == 1) {
                    duck = a->delete_duck(i);
                    b->add_duck(duck);
                    cout << "Да, сегодня мигрирую на какое-нибудь озеро (" << duck->is_name() << " мигрирует на Кратерное озеро)" << endl;
                    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
                    duck->migration_end();
                    i--;
                    Sleep(1000);
                }
                else {
                    c->add_duck(duck);
                    duck = a->delete_duck(i);
                    cout << "Да, сегодня мигрирую на какое-нибудь озеро (" << duck->is_name() << " мигрирует на озеро Мьоса)" << endl;
                    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
                    duck->migration_end();
                    i--;
                    Sleep(1000);
                }
            }
        }
        for (int i = 1; i <= b->get_size(); i++) {
            duck = b->choose_duck(i);
            if ((duck) && (duck->is_migration() == 1)) {
                cout << " Уточка по имени " << duck->is_name() << " подумывает насчёт миграции" << endl;
                Sleep(1000);
```

```
42.                 if (rand() % 3 == 0) {
43.                     duck = b->delete_duck(i);
44.                     a->add_duck(duck);
45.                     cout << "Да, сегодня мигрирую на какое-нибудь
   озеро (" << duck->is_name() << " мигрирует на озеро Кива)" << endl;
46.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
47.                     duck->migration_end();
48.                     i--;
49.                     Sleep(1000);
50.                 }
51.                 else if (rand() % 3 == 1) {
52.                     cout << "Нет, сегодня плохой день для миграции ("
   << duck->is_name() << " остается)" << endl;
53.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
54.                     duck->migration_end();
55.                     Sleep(1000);
56.                 }
57.                 else {
58.                     c->add_duck(duck);
59.                     duck = b->delete_duck(i);
60.                     cout << "Да, сегодня мигрирую на какое-нибудь
   озеро (" << duck->is_name() << " мигрирует на озеро Мьоса)" << endl;
61.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
62.                     duck->migration_end();
63.                     i--;
64.                     Sleep(1000);
65.                 }
66.             }
67.         }
68.         for (int i = 1; i <= c->get_size(); i++) {
69.             duck = c->choose_duck(i);
70.             if ((duck) && (duck->is_migration() == 1)) {
71.                 cout << " Уточка по имени " << duck->is_name() << "
   подумывает насчёт миграции" << endl;
72.                 Sleep(1000);
73.                 if (rand() % 3 == 0) {
```

```
74.                     duck = c->delete_duck(i);
75.                     a->add_duck(duck);
76.                     cout << "Да, сегодня мигрирую на какое-нибудь
   озеро (" << duck->is_name() << " мигрирует на озеро Кива)" << endl;
77.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
78.                     i--;
79.                     Sleep(1000);
80.                 }
81.                 else if (rand() % 3 == 1) {
82.                     duck = c->delete_duck(i);
83.                     b->add_duck(duck);
84.                     cout << "Да, сегодня мигрирую на какое-нибудь
   озеро (" << duck->is_name() << " мигрирует на Кратерное озеро)" <<
   endl;
85.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
86.                     i--;
87.                     Sleep(1000);
88.                 }
89.                 else {
90.                     cout << "Нет, сегодня плохой день для миграции ("
   << duck->is_name() << " остается)" << endl;
91.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
92.                     Sleep(1000);
93.                 }
94.                 duck->migration_end();
95.             }
96.         }
97.         for (int i = 1; i <= a->get_size(); i++) {
98.             duck = a->choose_duck(i);
99.             if (duck->is_migration() == 2) {
100.                duck->migration_again();
101.            }
102.        }
103.        for (int i = 1; i <= b->get_size(); i++) {
104.            duck = b->choose_duck(i);
105.            if (duck->is_migration() == 2) {
```

```
                duck->migration_again();
            }
        }
        for (int i = 1; i <= c->get_size(); i++) {
            duck = c->choose_duck(i);
            if (duck->is_migration() == 2) {
                duck->migration_again();
            }
        }
        Sleep(5000);
        cout << "Мирация окончена!" << endl;
        system("cls");
    }
```

Приложение 4

```
void migration(Ozero* a, Ozero* b, Ozero* c) {
    MamaCryak* duck;
    cout << "Начинается стадия миграции!" << endl;
    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
    Sleep(2000);
    for (int i = 1; i <= a->get_size(); i++) {
        duck = a->choose_duck(i);
        if ((duck) && (duck->is_migration() == 1)) {
            cout << " Уточка по имени " << duck->is_name() << " подумывает насчёт миграции" << endl;
                Sleep(1000);
                if (rand() % 3 == 0) {
                    cout << "Нет, сегодня плохой день для миграции (" << duck->is_name() << " остается)" << endl;
                    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
                    duck->migration_end();
                    Sleep(1000);
                }
                else if (rand() % 3 == 1) {
                    duck = a->delete_duck(i);
```

```
19.                     b->add_duck(duck);
20.                     cout << "Да, сегодня мигрирую на какое-нибудь
   озеро (" << duck->is_name() << " мигрирует на Кратерное озеро)" <<
   endl;
21.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
22.                     duck->migration_end();
23.                     i--;
24.                     Sleep(1000);
25.                 }
26.                 else {
27.                     c->add_duck(duck);
28.                     duck = a->delete_duck(i);
29.                     cout << "Да, сегодня мигрирую на какое-нибудь
   озеро (" << duck->is_name() << " мигрирует на озеро Мьоса)" << endl;
30.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
31.                     duck->migration_end();
32.                     i--;
33.                     Sleep(1000);
34.                 }
35.             }
36.         }
37.         for (int i = 1; i <= b->get_size(); i++) {
38.             duck = b->choose_duck(i);
39.             if ((duck) && (duck->is_migration() == 1)) {
40.                 cout << " Уточка по имени " << duck->is_name() << "
   подумывает насчёт миграции" << endl;
41.                 Sleep(1000);
42.                 if (rand() % 3 == 0) {
43.                     duck = b->delete_duck(i);
44.                     a->add_duck(duck);
45.                     cout << "Да, сегодня мигрирую на какое-нибудь
   озеро (" << duck->is_name() << " мигрирует на озеро Кива)" << endl;
46.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
47.                     duck->migration_end();
48.                     i--;
49.                     Sleep(1000);
```

```
                }
                else if (rand() % 3 == 1) {
                    cout << "Нет, сегодня плохой день для миграции (" << duck->is_name() << " остается)" << endl;
                    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
                    duck->migration_end();
                    Sleep(1000);
                }
                else {
                    c->add_duck(duck);
                    duck = b->delete_duck(i);
                    cout << "Да, сегодня мигрирую на какое-нибудь озеро (" << duck->is_name() << " мигрирует на озеро Мьоса)" << endl;
                    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
                    duck->migration_end();
                    i--;
                    Sleep(1000);
                }
            }
        }
        for (int i = 1; i <= c->get_size(); i++) {
            duck = c->choose_duck(i);
            if ((duck) && (duck->is_migration() == 1)) {
                cout << " Уточка по имени " << duck->is_name() << " подумывает насчёт миграции" << endl;
                Sleep(1000);
                if (rand() % 3 == 0) {
                    duck = c->delete_duck(i);
                    a->add_duck(duck);
                    cout << "Да, сегодня мигрирую на какое-нибудь озеро (" << duck->is_name() << " мигрирует на озеро Кива)" << endl;
                    cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
                    i--;
                    Sleep(1000);
                }
                else if (rand() % 3 == 1) {
```

```
82.                     duck = c->delete_duck(i);
83.                     b->add_duck(duck);
84.                     cout << "Да, сегодня мигрирую на какое-нибудь
   озеро (" << duck->is_name() << " мигрирует на Кратерное озеро)" <<
   endl;
85.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
86.                     i--;
87.                     Sleep(1000);
88.                 }
89.                 else {
90.                     cout << "Нет, сегодня плохой день для миграции ("
   << duck->is_name() << " остается)" << endl;
91.                     cout <<
   "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;
92.                     Sleep(1000);
93.                 }
94.                 duck->migration_end();
95.             }
96.         }
97.         for (int i = 1; i <= a->get_size(); i++) {
98.             duck = a->choose_duck(i);
99.             if (duck->is_migration() == 2) {
100.                duck->migration_again();
101.            }
102.        }
103.        for (int i = 1; i <= b->get_size(); i++) {
104.            duck = b->choose_duck(i);
105.            if (duck->is_migration() == 2) {
106.                duck->migration_again();
107.            }
108.        }
109.        for (int i = 1; i <= c->get_size(); i++) {
110.            duck = c->choose_duck(i);
111.            if (duck->is_migration() == 2) {
112.                duck->migration_again();
113.            }
114.        }
115.        Sleep(5000);
```

```
116.         cout << "Мирация окончена!" << endl;
117.         system("cls");
118.     }
```

Приложение 5

```
1.while (get != 0) {
2.             menu(day);
3.             cin >> get;
4.             switch (get) {
5.             case (1):
6.                 system("cls");
7.                 cout << "Какая(ое) именно озеро/ферма вас интересует?"
   << endl;
8.                 cout << " 1 - озеро Киву " << endl;
9.                 cout << " 2 - Кратерное озеро " << endl;
10.                     cout << " 3 - озеро Мьоса " << endl;
11.                     cout << " 4 - ферма СОВИМОТ " << endl;
12.                     cin >> get;
13.                     switch (get) {
14.                     case (1):
15.                         system("cls");
16.                         if (kivu.get_size() != 0) {
17.                             cout << "На озере Киву проживают утки в
   количетве " << kivu.get_size() << ". Какую хотите опросить?" << endl;
18.                             do
19.                             {
20.                                 cout << "Пожалуйста, для того, чтобы
   обратиться к утке, введиите конктретное значение в промежутке от 1 до "
   << kivu.get_size() << " включительно" << endl;
21.                                 cin >> get;
22.                             } while (get > kivu.get_size() || get <
   1);
23.                             cout << endl;
24.                             a = kivu.choose_duck(get);
25.                             a->get_info();
26.                             cout << endl;
27.                         }
```

```
28.                         else {
29.                             cout << "На озере Киву, к сожалению, не
   осталось уток" << endl;
30.                         }
31.                         cout << " 1 - Вернуться в главное меню" <<
   endl;
32.                         cin >> get;
33.                         switch (get) {
34.                         case(1):
35.                             system("cls");
36.                             break;
37.                         }
38.                         break;
39.                     case (2):
40.                         system("cls");
41.                         if (kraternoe_ozero.get_size() != 0) {
42.                             cout << "На Кратерном озере проживают утки
   в количетве " << kraternoe_ozero.get_size() << ". Какую хотите
   опросить?" << endl;
43.                             do
44.                             {
45.                                 cout << "Пожалуйста, для того, чтобы
   обратиться к утке, введиите конктретное значение в промежутке от 1 до "
   << kraternoe_ozero.get_size() << " включительно" << endl;
46.                                 cin >> get;
47.                             } while (get > kraternoe_ozero.get_size()
   || get < 1);
48.                             cout << endl;
49.                             a = kraternoe_ozero.choose_duck(get);
50.                             a->get_info();
51.                             cout << endl;
52.                         }
53.                         else {
54.                             cout << "На Кратерном озере, к сожалению,
   не осталось уток" << endl;
55.                         }
56.                         cout << " 1 - Вернуться в главное меню" <<
   endl;
57.                         cin >> get;
```

```
58.                             switch (get) {
59.                             case(1):
60.                                 system("cls");
61.                                 break;
62.                             }
63.                             break;
64.                         case (3):
65.                             system("cls");
66.                             if (mjosa.get_size() != 0) {
67.                                 cout << "На озере Мьоса проживают утки в количетве " << mjosa.get_size() << ". Какую хотите опросить?" << endl;
68.                                 do
69.                                 {
70.                                     cout << "Пожалуйста, для того, чтобы обратиться к утке, введиите конктретное значение в промежутке от 1 до " << mjosa.get_size() << " включительно" << endl;
71.                                     cin >> get;
72.                                 } while (get > mjosa.get_size() || get < 1);
73.                                 cout << endl;
74.                                 a = mjosa.choose_duck(get);
75.                                 a->get_info();
76.                                 cout << endl;
77.                             }
78.                             else {
79.                                 cout << "На озере Мьоса, к сожалению, не осталось уток" << endl;
80.                             }
81.                             cout << " 1 - Вернуться в главное меню" << endl;
82.                             cin >> get;
83.                             switch (get) {
84.                             case(1):
85.                                 system("cls");
86.                                 break;
87.                             }
88.                             break;
89.                         case (4):
90.                             system("cls");
```

```
91.                         if (COBUMOT.get_size() != 0) {
92.                             cout << "На ферме COBUMOT проживают утки в количетве " << COBUMOT.get_size() << ". Какую хотите опросить?" << endl;
93.                             do
94.                             {
95.                                 cout << "Пожалуйста, для того, чтобы обратиться к утке, введиите конктретное значение в промежутке от 1 до " << COBUMOT.get_size() << " включительно" << endl;
96.                                 cin >> get;
97.                             } while (get > COBUMOT.get_size() || get < 1);
98.                             cout << endl;
99.                             a = COBUMOT.choose_duck(get);
100.                            a->get_info();
101.                            cout << endl;
102.                        }
103.                        else {
104.                            cout << "На ферме COBUMOT, к сожалению, пока нет уток" << endl;
105.                        }
106.                        cout << " 1 - Вернуться в главное меню" << endl;
107.                        cin >> get;
108.                        switch (get) {
109.                        case(1):
110.                            system("cls");
111.                            break;
112.                        }
113.                    }
114.                    break;
115.                case (2):
116.                    system("cls");
117.                    cout << "Какая(ое) именно озеро/ферма вас интересует?" << endl;
118.                    cout << " 1 - озеро Киву " << endl;
119.                    cout << " 2 - Кратерное озеро " << endl;
120.                    cout << " 3 - озеро Мьоса " << endl;
121.                    cout << " 4 - ферма COBUMOT " << endl;
```

```
122.                     cin >> get;
123.                     switch (get) {
124.                     case(1):
125.                         system("cls");
126.                         cout << "Информация по озеру Киву:" << endl;
127.                         kivu.get_information();
128.                         cout << " 1 - Вернуться в главное меню" <<
    endl;
129.                         cin >> get;
130.                         switch (get) {
131.                         case(1):
132.                             system("cls");
133.                             break;
134.                         }
135.                         break;
136.                     case(2):
137.                         system("cls");
138.                         cout << "Информация по Кратерному озеру:" <<
    endl;
139.                         kraternoe_ozero.get_information();
140.                         cout << " 1 - Вернуться в главное меню" <<
    endl;
141.                         cin >> get;
142.                         switch (get) {
143.                         case(1):
144.                             system("cls");
145.                             break;
146.                         }
147.                         break;
148.                     case(3):
149.                         system("cls");
150.                         cout << "Информация по озеру Мьоса:" << endl;
151.                         mjosa.get_information();
152.                         cout << " 1 - Вернуться в главное меню" <<
    endl;
153.                         cin >> get;
154.                         switch (get) {
155.                         case(1):
156.                             system("cls");
```

```
157.                             break;
158.                         }
159.                         break;
160.                     case(4):
161.                         system("cls");
162.                         cout << "Информация по ферме CUBMOUT:" <<
    endl;
163.                         COBUMOT.get_information();
164.                         cout << " 1 - Вернуться в главное меню" <<
    endl;
165.                         cin >> get;
166.                         switch (get) {
167.                         case(1):
168.                             system("cls");
169.                             break;
170.                         }
171.                         break;
172.                     }
173.                     break;
174.                 }
175.             }
```