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

Институт кибербезопасности и цифровых технологий
Кафедра КБ-14 «Цифровые технологии обработки данных»

ОТЧЁТ ПО УЧЕБНОЙ ПРАКТИКЕ

Ознакомительная практика

Тема практики: «Создание информационной системы на языке C#»

приказ Университета о направлении на практику от «14» января 2022 г. № 49-У

Отчет представлен к рассмотрению:

Студент группы: «__» ____ 2022 г. _____ (Подпись) Азбукин Д.Ю.

Отчет утвержден.
Допущен к защите:

Руководитель практики от кафедры «__» ____ 2022 г. _____ (Подпись) Сачков В.Е.

Москва 2022 г.

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

Институт кибербезопасности и цифровых технологий
Кафедра КБ-14 «Цифровые технологии обработки данных»

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ НА УЧЕБНУЮ ПРАКТИКУ
Ознакомительная практика

Студенту 1 курса учебной группы БСБО-10-21
Азбукину Даниилу Юрьевичу

Место и время практики: РТУ МИРЭА, кафедра КБ-14 «Цифровые технологии обработки данных», с «10» февраля 2022 г. по «30» мая 2022 г.

Должность на практике: студент

1. ЦЕЛЕВАЯ УСТАНОВКА: развитие способностей в области анализа и моделирования прикладных процессов с учетом выбранной темы исследования

2. СОДЕРЖАНИЕ ПРАКТИКИ:

2.1. Изучить: исследовать информационные и прикладные процессы

2.2. Практически выполнить: применить современные инструментальные средства для моделирования информационных и прикладных процессов

2.3. Ознакомиться: с уровнем развития информационных и прикладных процессов с учетом темы исследования

3. ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ: оформить отчет

4. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ УКАЗАНИЯ: в процессе практики рекомендуется использовать издания и отраслевую литературу годом издания не старше 5 лет

Заведующий кафедрой:

«10» февраля 2022 г. _____ (Иванова И.А)
(подпись)

СОГЛАСОВАНО:
Руководитель практики от кафедры
«10» февраля 2022 г. _____ Сачков В.Е.

(подпись)

Задание получил
«10» февраля 2022 г. _____ Азбукин Д.Ю.

(подпись)

Проведенные инструктажи:

Охрана труда: «10» февраля 2022 г.

Инструктирующий _____ (Сачков В.Е., доцент, к.т.н)
(подпись)

Инструктируемый _____ Азбукин Д.Ю.
(подпись)

Техника безопасности: «10» февраля 2022 г.

Инструктирующий _____ (Сачков В.Е., доцент, к.т.н)
(подпись)

Инструктируемый _____ Азбукин Д.Ю.
(подпись)

Пожарная безопасность: «10» февраля 2022 г.

Инструктирующий _____ (Сачков В.Е., доцент, к.т.н)
(подпись)

Инструктируемый _____ Азбукин Д.Ю.
(подпись)

С правилами внутреннего распорядка ознакомлен: «10» февраля 2022 г.

_____ Азбукин Д.Ю.
(подпись)

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

РАБОЧИЙ ГРАФИК ПРОВЕДЕНИЯ УЧЕБНОЙ ПРАКТИКИ

студента Азбукина Даниила Юрьевича 1 курса группы БСБО-10-21 очной формы обучения, обучающегося по направлению 09.03.02 Информационные системы и технологии профиль «Технологии визуального программирования»

Неделя	Сроки выполнения	Этап	Отметка о выполнении
1	«10» февраля 2022 г.- «10» марта 2022 г.	Подготовительный этап, включающий в себя организационное собрание (Вводная лекция о порядке организации и прохождения учебной практики, инструктаж по технике безопасности)	
2	«11» марта 2022 г. – «30» апреля 2022 г.	Выполнение задания по практике в соответствии с выданным заданием студента. (Мероприятия по сбору, обработке и структурированию материала, выполнение поставленной задачи)	
3	«01» мая 2022 г. – «24» мая 2022 г.	Подготовка отчета по практике (Оформление материалов отчета в полном соответствии с требованиями на оформление учебных работ студентов)	
4	«25» мая 2022 г.	Защита отчета по учебной практике у руководителя практики. (Представление отчета по практике к защите)	

Согласовано:

Заведующий кафедрой _____/ Иванова И.А., к.т.н., доцент/

Руководитель практики от кафедры _____/ Сачков В.Е., доцент, к.т.н /

Обучающийся _____/Азбукин Д.Ю./

ОТЧЁТ

по учебной практике

студента 1 курса учебной группы БСБО-10-21 института ИКБ

Азбукина Даниила Юрьевича

1. Практику проходил с 10.02.2022 г. по 30.05.2022 г. в ФГБОУ ВО «МИРЭА – Российский технологический университет», на кафедре КБ-14 «Цифровые технологии обработки данных», студент

(место прохождения практики и должность)

2. Задание на практику выполнил

в полном объеме

(указать: в полном объеме или частично)

Не выполнены следующие задания:

(указать также причины невыполнения)

Подробное содержание выполненной на практике работы и достигнутые результаты: проведено исследование прикладной области в части изучения

Предложения по совершенствованию организации и прохождения практики:
предложений нет

Студент _____ (Азбукин Д.Ю.)

(подпись)

«__» _____ 20___ г

Заключение руководителя практики

Приобрел следующие профессиональные навыки: студент продемонстрировал профессиональные умения и навыки, знание и понимание прикладной области, задач, требующих решения в прикладной области, современные подходы и средства решения прикладных задач разных классов, умение находить и работать с различными источниками информации по профессиональной деятельности, структурировать отчет с учетом тематики исследования

Проявил себя как: дисциплинированный ответственный специалист: соблюдал сроки календарного графика практики, регулярно отчитывался о проделанных этапах работ; за срок прохождения практики не получил ни одного замечания - проявляет инициативу, четко и в определенные сроки выполняет задания; в любой ситуации уважителен в общении с другими.

«__» _____ 20__ г

Отчет проверил:
Руководитель практики от Университета

_____(Сачков В.Е.)

(подпись)

Содержание

Задание

Вариант 18.

ДО ЗАСУХИ: 14 дней

Колония <рыжие> популяция: рабочих=14, воинов=8, особенных=1;

Королева <Шарлотта>(здоровье=17, защита=8, урон=29), цикл роста личинок 2-3 дней, может создать 1-5 королев;

- <обычный> РАБОЧИЙ(здоровье=1, защита=0) может брать 1 ресурс: 'веточка' за раз.

- <старший> РАБОЧИЙ(здоровье=2, защита=1) может брать 1 ресурс: 'веточка или веточка' за раз.

- <старший неповторимый> РАБОЧИЙ(здоровье=2, защита=1) может брать 1 ресурс: 'камушек или росинка' за раз; полностью неуязвим для всех атак (даже смертельных для неуязвимых), игнорирует все модификаторы врагов, всегда находит нужный ресурс в куче, даже если его больше нет.

- <обычный> ВОИН(здоровье=1, защита=0, урон=1) может атаковать 1 цель за раз и наносит 1 укус.

- <продвинутый> ВОИН(здоровье=6, защита=2, урон=4) может атаковать 2 цели за раз и наносит 1 укус.

- <продвинутый мстительный> ВОИН(здоровье=6, защита=2, урон=4) может атаковать 2 цели за раз и наносит 1 укус; убивает своего убийцу, даже если он неуязвим.

Особенное насекомое <ленивый обычный агрессивный настойчивый - Шмель>(здоровье=25, защита=6, урон=8): не может брать ресурсы; может быть атакован войнами; атакует врагов(2 цели за раз и наносит 1 укус); всегда наносит укус, даже если был убит.

Колония <черные> популяция: рабочих=15, воинов=5, особенных=1;

Королева <Шарлотта>(здоровье=16, защита=9, урон=23), цикл роста личинок 3-4 дней, может создать 2-5 королев;

- <легендарный> РАБОЧИЙ(здоровье=10, защита=6) может брать 3 ресурса: 'листик и камушек и росинка' за раз.

- <обычный> РАБОЧИЙ(здоровье=1, защита=0) может брать 1 ресурс: 'веточка' за раз.

- <легендарный спринтер> РАБОЧИЙ(здоровье=10, защита=6) может брать 3 ресурса: 'листик и веточка и росинка' за раз; не может быть атакован первым.

- <старший> ВОИН(здоровье=2, защита=1, урон=2) может атаковать 1 цель за раз и наносит 1 укус.

- <элитный> ВОИН(здоровье=8, защита=4, урон=3) может атаковать 2 цели за раз и наносит 2 укуса.

- <легендарный толстый> ВОИН(здоровье=10, защита=6, урон=4) может атаковать 3 цели за раз и наносит 1 укус; принимает все атаки на себя, здоровье и защита увеличены в двое.

Особенное насекомое <ленивый неуязвимый агрессивный точный - Толстоножка>(здоровье=25, защита=6, урон=8): не может брать ресурсы; не может быть атакован войнами; атакует врагов(2 цели за раз и наносит 3 укуса); игнорирует защиту и может наносить урон неуязвимым насекомым.

Список Куч:

Куча 1 ресурсы: веточка: 10; листик: 17; камушек: 31; росинка: 45;

Куча 2 ресурсы: веточка: 12; листик: 18; росинка: 29;

Куча 3 ресурсы: веточка: 33; листик: 18; камушек: 23;

Куча 4 ресурсы: веточка: 29; росинка: 15;

Куча 5 ресурсы: веточка: 13; листик: 34; росинка: 39;

ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ

Появляется <Белка> на '7' дней с эффектом (происходит каждый день): ворует половину веточек со случайной кучи.

Реализация задания

Задание выполнено на языке программирования C#, в среде разработки JetBrains Rider. Для его реализации потребовалось подключение двух пространств имен – System и System.Collections.Generic. Исходный код см. в Приложении.

Исходя из условия задания, все муравьи обладают перечнем общих для друг друга характеристик и действий. Поэтому выгодно создать класс, от которого будут наследоваться другие классы муравьев, описывающий эти общие характеристики как поля, и методы, такие как здоровье, урон, защита, имя, флаг о том что муравей живой и класс колонии, а также метод получении урона. Данный класс наследуется от интерфейса ITakeDamage, который используется для обработки получения урона. Данный класс и интерфейс представлен в Листинге 1.

Листинг 1 – Создание основного класса и интерфейса

```
public class Ant:ITakeDamage
    {
        public int hp;
        public int def;
        public int damage;
        public string name;
        public bool IsAlive;
        public Colony _colony;
        public Ant(string name, int hp, int def, int damage,Colony _colony)
        {
            this.hp = hp;
            this.def = def;
            this.damage = damage;
            this.name = name;
            IsAlive = true;
            this._colony = _colony;
        }
        public virtual void TakeDamage(int dam,bool ignoreDef)
        {
            IsAlive = false;
        }
      }
public interface ITakeDamage
    {
        void TakeDamage(int dam,bool ignoreDef);
    }
```

Перед рассмотрением классов воинов, рабочих и т.д. следует рассмотреть класс Resourses – Листинг 2, в данном решении он имеет два массива, первый

отвечает за тип ресурсов (веточка, листик, камень, росинка), а второй отвечает за их количество, в указанном ранее порядке.

Листинг 2 – Создание класса Resourses

```
public class Resourses
    {
        public Resources[] typeOfRes;  //порядок: веточка -> листик -> камень ->капля
        public int[] amountOfRes;
        public Resourses(int branch,int leaf,int stone,int drop)
        {
            this.typeOfRes = new Resources[4] {Resources.Branch, Resources.Leaf,
Resources.Stone, Resources.Drop};
            this.amountOfRes = new int[4] {branch, leaf, stone, drop};
        }
    }
    public enum Resources
    {
        Branch,
        Stone,
        Drop,
        Leaf
    }
```

Класс рабочих – Worker, наследуется от основного класса Ant, и интерфейса ICanGoToHeap, данный интерфейс используется для походов на кучу (об данном интерфейсе далее). Более того, класс Worker имеет поля класса Resourses, чтобы обозначить ресурсы, которые он может собирать по условию задачи, и собранные ресурсы на куче, также он имеет поле, отвечающее за принцип сбора ресурсов (некоторые муравьи могут брать один из видов ресурсов, а некоторые берут несколько одновременно). Класс имеет поля: типа рабочих enum, класс кучи на которую он распределен и массив модификаторов. Класс содержит методы: распределения на кучу, удаления при смерти, вывода информации, сбора ресурсов, для интерфейсов.

Класс рабочих – Wariour, наследуется от класса Ant, и интерфейсов IAttack и ICanGoToHeap, и имеет поля: типа муравья, количества целей, количества укусов, класса кучи, на которую он был распределен, массив модификаторов, и количества оставшихся атак на куче. Также содержит методы: получения урона, распределения на кучу, удаления при смерти, вывода информации и методы для интерфейсов. Метод получения урона (Листинг 3) у воинов принимает два значения, первое – урон, второе – булевая переменная

отвечающая за игнорирование защиты (у некоторых типов муравьев есть модификатор игнорирования защита), если игнорирование защиты существует, то урон отнимается от здоровья. Если же игнорирования защиты нет, то существует 3 случая: 1 – защита больше атаки – отнимается 1 здоровье, 2 – защита равна урону – от здоровья отнимается половина урона, 3 – защита меньше урона – от здоровья отнимается урон минус половина защиты.

Листинг 3 – Система получения урона у воинов

```
public override void TakeDamage(int dam, bool ignoreDef)
        {
            if (ignoreDef)
            {
                hp -= dam;
                if (hp <= 0)
                {
                    IsAlive = false;
                }
            }
            else
            {
                if (dam < def)
                {
                    hp -= 1;
                }
                else if (dam == def)
                {
                    hp -= dam / 2;
                }
                else
                {
                    hp -= dam - def / 2;
                }

                if (hp <= 0)
                {
                    IsAlive = false;
                    DayController.heapGoing -= GoToHeap;
                }
            }
        }
```

Класс особого насекомого не отличается от класса Wariour, отличается только вывод информации, потому что в данном решении особое насекомое – это воин имеющий другие модификаторы.

За колонии отвечает класс Colony (Листинг 4), который содержит поля класса королевы, массив класса воинов, массив класса рабочих, и массив класса особого насекомого, также он содержит поля названия колонии, типа колонии,

массив класса ресурсов, и массив класса колоний – дружественных колоний. Данный класс содержит метод по удалению мертвых муравьев из колонии, по получению ресурсов от рабочих, которые вернулись с кучи, более того, содержит метод, который подсчитывает все ресурсы и возвращает это значение (это создано для того, чтобы выявить выжившую колонию после засухи), и метод для вывода информации о колонии.

Листинг 4 – Создание класса колонии Colony без вывода информации о колонии

```
public class Colony
    {
        //----------------Юниты--------------
        public Queen Queen;
        public List<Wariour> units_Wariour;
        public List<Worker> units_Worker;
        public List<SpecialInsect> SpecialInsect;
        //--------------Значения колонии-------------
        public string name;
        public ColonyTypes ColonyType;
        public Resourses ColonyRes;
        public List<Colony> FriendlyColonies;
        //--------------------Конструктор-------------------
        public Colony()
        {
            this.units_Wariour = new List<Wariour>();
            this.units_Worker = new List<Worker>();
            this.SpecialInsect = new List<SpecialInsect>();
            this.ColonyRes = new Resourses(0, 0, 0, 0);
            DayController.screen_1 += print_Screen_1;
            DayController.screen_2 += print_Screen_2;
            DayController.screen_3_HeakStart += Screen_3_HeakStart;
            DayController.DeleteDead_Borning_print += DelDeadPrintAnts;
            DayController.DeleteDead_Borning_print += GetResourses_BornInsects;
            FriendlyColonies = new List<Colony>();
        }
        //------------------------Удаление мертвых муравьев-------------------
        public void DelDeadPrintAnts(int day)
        {
            int deadWariours = 0;
            int deadWorkers = 0;
            int deadSpecIns = 0;
            for (int i = 0; i < units_Wariour.Count; i++)
            {
                if (!(units_Wariour[i].IsAlive))
                {
                    units_Wariour.RemoveAt(i);
                    deadWariours++;
                }
            }

            for (int i = 0; i < units_Worker.Count; i++)
            {
                if (!(units_Worker[i].IsAlive))
                {
                    units_Worker.RemoveAt(i);
                    deadWorkers++;
```

```
                }
            }
            for (int i = 0; i < SpecialInsect.Count; i++)
            {
                if (SpecialInsect[i].IsAlive == false)
                {
                    SpecialInsect.RemoveAt(i);
                    deadSpecIns++;
                }
            }

            Console.WriteLine($"С колонии {name} вернулись: \n" +
                              $"---р={units_Worker.Count}, в={units_Wariour.Count},
o={SpecialInsect.Count} \n" +
                              $"---Потери: р={deadWorkers}, в={deadWariours},
o={deadSpecIns}");
        }

}
```

Создание класса королевы (Листинг 5) во много отличается от создания классов воинов, рабочих и особых насекомых, потому что королева не ходит на кучи и должна создавать новых королев и муравьев. Класс королев содержит поля цикла роста личинок, количества возможных королев, массивов типов рабочих и воинов, дат создания личинок (с муравьями), количества личинок, и дней до рождения, а также количество рожденных королев. Этот класс содержит метод рождения муравьев, он принимает в себя день, если личинок нет, то создается случайное количество личинок от 2 до 10, назначается в переменную DayOfEggBirth значение дня, назначается случайное количество дней до рождения полноценных муравьев из цикла роста личинок. Однако, если личинки уже созданы, метод проверяет дни до рождения из личинок муравьев, если время совпадает, то с помощью цикла создается на случайной основе муравей и добавляется в массив муравьев колонии, тип которого выбирается случайно в зависимости от типа колонии, в которой состоит королева, после чего снова создаются личинки. Также, метод проверяет количество созданных королев (чтобы их количество не превысило допустимое значение), если рождается королева, то с определенной вероятностью создается новая колония с королевой (данная королева не может создавать королев), которая добавляется в массив FriendlyColony в колонии матери, и функционирует как отдельная колония.

Листинг 5 – Создание класса королевы

```
public class Queen:Ant
    {
        public int[] Borning;
        public int[] Queens;
        public List<WariourType> wariousType;
        public List<WorkerType> workersType;
        public int DayOfEggBirth;
        public int eggs;
        public int BorningTime;
        public int BornedQueens;
        public Queen(string name, int hp, int def, int damage,Colony _colony, int[]
borning, int[] queens,List<WariourType> wariousType,List<WorkerType> workersType) :
base(name,hp, def, damage,_colony)
        {
            this.wariousType = wariousType;
            this.workersType = workersType;
            Borning = borning;
            Queens = queens;
            eggs = 0;
            BornedQueens = 0;
        }
        //-------------------Рождение муравьев------------------
        public void BornInsects(int Day)
        {
            if (eggs == 0)
            {
                eggs = Globals._random.Next(2, 10);
                DayOfEggBirth = Day;
                BorningTime = Globals._random.Next(Borning[0], Borning[1]+1);
                Console.WriteLine($"---Новые личинки: {eggs} (еще {BorningTime-
Day+DayOfEggBirth} дней)");
            }
            else if (Day == (DayOfEggBirth + BorningTime))
            {
                int newWorkers = 0;
                int newWariours = 0;
                int newQueen = 0;
                int CanBornQueens = 3;
                while (eggs != 0)
                {
                    if (BornedQueens >= Queens[1])
                    {
                        CanBornQueens = 2;
                    }
                    switch (Globals._random.Next(0, CanBornQueens))
                    {
                        case 0:
                        {
                            WariourType tmp = wariousType[Globals._random.Next(0,
wariousType.Count)];
                            switch (tmp)
                            {
                                case WariourType.USUAL:
_colony.units_Wariour.Add(Globals.Create_Usual_Wariour(_colony));
                                    break;
                                case WariourType.ELITE:
_colony.units_Wariour.Add(Globals.Create_Elite_Wariour(_colony));
                                    break;
                                case WariourType.OLDER:
_colony.units_Wariour.Add(Globals.Create_Older_Wariour(_colony));
                                    break;
```

```
                                case WariourType.ADVANCED:
_colony.units_Wariour.Add(Globals.Create_Advanced_Wariour(_colony));
                                    break;
                                case WariourType.LEGENDARY_FAT:
_colony.units_Wariour.Add(Globals.Create_LegendaryFat_Wariour(_colony));
                                    break;
                                case WariourType.ADVANCED_REVENGEFULL:
_colony.units_Wariour.Add(Globals.Create_AdvancedRevengefull_Wariour(_colony));
                                    break;
                            }

                            newWariours++;
                            break;
                        }
                        case 1:
                        {
                            WorkerType tmp = workersType[Globals._random.Next(0,
workersType.Count)];
                            switch (tmp)
                            {
                                case WorkerType.OLDER:
_colony.units_Worker.Add(Globals.Create_Older_Worker(_colony));
                                    break;
                                case WorkerType.USUAL:
_colony.units_Worker.Add(Globals.Create_Usual_Worker(_colony));
                                    break;
                                case WorkerType.LEGENDARY:
_colony.units_Worker.Add(Globals.Create_Legendary_Worker(_colony));
                                    break;
                                case WorkerType.OLDER_UNIQUE:
_colony.units_Worker.Add(Globals.Create_OlderUnique_Worker(_colony));
                                    break;
                                case WorkerType.LEGENDARY_RUNNER:
_colony.units_Worker.Add(Globals.Create_LegendaryRunner_Worker(_colony));
                                    break;
                            }
                            newWorkers++;
                            break;
                        }
                        case 2:
                        {
                            BornedQueens++;
                            int tmp = Globals._random.Next(0, 2);
                            if ((_colony.FriendlyColonies.Count <
Queens[1])&&(tmp==0))
                            {
                                Globals.FriendlyColony_Creation(_colony);
                            }
                            newQueen++;
                            break;
                        }
                    }
                    eggs--;
                }
                eggs = Globals._random.Next(2, 10);
                DayOfEggBirth = Day;
                BorningTime = Globals._random.Next(Borning[0], Borning[1]+1);
                Console.WriteLine($"---Выросли: р={newWorkers}, в={newWariours},
к={newQueen} \n" +
                                  $"---Новые личинки: {eggs} (еще {BorningTime-
Day+DayOfEggBirth} дней)");
            }
            else
```

```
            {
                Console.WriteLine($"---Личинки еще растут({BorningTime-
Day+DayOfEggBirth})");
            }
        }
    }
}
```

Для инициализации начальных параметров и многих функций для создания муравьев существует отдельный статический класс Globals. Он содержит методы инициализации: классов муравьев по каждым типам, начальных двух королев, начальных двух колоний, начальных ресурсов кучи; имеет методы вывода информации по каждому типу муравья и по модификаторам, и методы создания колоний для родившихся королев.

За кучи также отвечает отдельный класс Heap, который наследуется от класса Resourses, так как из него будут брать ресурсы рабочие. Этот класс содержит номер кучи и булевую переменную, которая отвечает за возможность похода на эту кучу, так как куча может быть истощена и муравьи на эту кучу ходить не будут, и содержит массив насекомых с интерфейсом ICanGoToHeap. Также, в классе Heap содержится система походов на кучу. Сначала, есть метод который добавляет всех, кого распределили в кучу. При истощении кучи существует метод, который меняет булевую переменную IsOpen на false и тогда насекомые не будут распределяться на эту кучу. В классе каждого насекомого, который может ходить на кучу, есть метод GoToHeap(), назначающий кучу, на которую они пойдут, и добавляющий их в массив насекомых на куче. Далее, метод, который размешивает массив, чтобы насекомые из разных куч шли в случайном порядке. И существует функция самого похода, в которой сначала размешивается массив, далее цикл проходится по всем рабочим и они собирают ресурсы в куче, далее, цикл проходится по массиву насекомых на куче, если насекомое наследуется от интерфейса IAttack, то в переменную записывается количество целей, которых он может атаковать, Далее создается массив, который включает в себя список возможных целей для атаки.

Существует метод который возвращает этот массив (Листинг 6), который принимает в себя массив насекомых на куче, насекомое и количество боев на куче, данный метод возвращает массив возможных целей для атаки, проверяя что атакующий и атакуемый не находятся в одной или дружественных колониях, а также проверяет все модификаторы, которые влияют на выбор насекомого в качестве цели.

Листинг 6 – Создание метода для возвращения возможных целей

```
public List<int> ExistenceOfEnemi(List<ICanGoToHeap> units,ICanGoToHeap wariour,int
amountOfFights)
        {
            List<int> Targets = new List<int>();
            for (int i = 0; i < units.Count; i++)
            {
                bool IsAgresive = true;
                foreach (Colony friendlyColony in
wariour.MyColony().FriendlyColonies)
                {
                    if (friendlyColony == units[i].MyColony())
                    {
                        IsAgresive = false;
                    }
                }
                if ((units[i].MyColony() != wariour.MyColony()) &&

(!units[i].ModificatorsReturn().Contains(Modificator.CantBeAttacked))&&

((!units[i].ModificatorsReturn().Contains(Modificator.CantBeAttackedFirst))||(units[i
].ModificatorsReturn().Contains(Modificator.CantBeAttackedFirst)&&amountOfFights!=0))
)
                {
                    Targets.Add(i);
                }
            }
            return Targets;
        }
```

После создания массива возможных целей проверяется существование насекомого, которое имеет модификатор провокация (враги на куче атакуют только его). Как только выбор цели завершен начинается бой, за бой отвечает статический класс Battle (Листинг 7), в котором есть функция Fight. Она принимает интерфейс IAttack и интерфейс ICanGoToHeap, и проверяет атакующего и атакуемого на модификатор игнорирования защиты. Далее у одного и второго вызывается функция TakeDamage, которая принимает атаку другого и булевую переменную о игнорировании защиты.

Листинг 7 – Создание боя

```
public static class Battle
    {
        public static void Fight(IAttack fir,ICanGoToHeap sec)
        {

            if (fir.ModificatorsReturn().Contains(Modificator.IgnoreDef))
            {
                IAttack temp = (IAttack) fir;
                sec.TakeDamage(temp.Attack(),true);
            }
            else
            {
                IAttack temp = (IAttack) fir;
                sec.TakeDamage(temp.Attack(),false);
            }

            if ((sec is
IAttack)&&(!fir.ModificatorsReturn().Contains(Modificator.CantBeAttacked)))
            {
                if (sec.ModificatorsReturn().Contains(Modificator.IgnoreDef))
                {
                    IAttack temp = (IAttack) sec;
                    fir.TakeDamage(temp.Attack(), true);
                }
                else
                {
                    IAttack temp = (IAttack) sec;
                    fir.TakeDamage(temp.Attack(), false);
                }
            }
        }
    }
```

Если кто-либо из участвующих в бое умирает, то проверяется модификатор «убивает своего убийцу». У мертвых вызывается функция смерти и добавляется модификатор «не может быть атакован», и обнуляется счетчик количества целей для атаки (в таком случае, никто не будет его атаковать и он никого не сможет атаковать), если у него нет модификатора «атакует даже после смерти». В конце каждого цикла атаки, из массива удаляются убитые насекомые.

Сами кучи хранятся в статическом классе Heaps (Листинг 8). В нем хранится массив куч и функция по удалению кучи при ее истощении, которая проверяет все ресурсы на куче и если количество ресурсов равно 0, то вызывается функция, закрывающая кучу. Данные для создания массива куч исходит из класса Globals.

Листинг 8 – Создание класса куч

```
public static class Heaps
    {
        public static List<Heap> heaps=Globals.Create_Heaps();
        public static void HeapDepletion()
        {
            for (int i = 0; i < heaps.Count;i++)
            {
                if ((heaps[i].amountOfRes[0] == 0) && (heaps[i].amountOfRes[1] == 0)
&&
                    (heaps[i].amountOfRes[2] == 0) && (heaps[i].amountOfRes[3] == 0))
                {
                    heaps[i].DeleteHeap();
                }
            }
        }
    }
```

Для контролирования последовательности действий и основной части программы используются события (event), которые хранятся в статическом классе DayController (Листинг 9). В нем событие, которое отвечает за вывод информации о всех насекомых, колониях, и общий вывод информации. Событие HeapGoing, использует функцию распределения на кучу у каждого муравья (который может ходить на кучу). Событие StartHeapEVENT начинает поход, запуская функцию Heak() у каждой из куч. Также, есть отдельное событие для удаления мертвых муравьев из колоний, получения ресурсов от рабочих, рождении новых муравьев и вывода информации о походе. При создании насекомых, куч и так далее, в конструкторе они добавляют определенные функции в разные события.

Листинг 9 – Инициализация событий

```
public static class DayController
    {
        public delegate void Screen_1_print();

        public static event Screen_1_print screen_1;

        public static void Screen_1()
        {
            screen_1.Invoke();
        }
        public delegate void Screen_2_print();

        public static event Screen_2_print screen_2;

        public static void Screen_2()
        {
            screen_2.Invoke();
```

```
        }
        public delegate void Screen_3_HeakStart();

        public static event Screen_3_HeakStart screen_3_HeakStart;

        public static void Screen_3_Start()
        {
            screen_3_HeakStart.Invoke();
        }
        //-------------GoToHeap------------
        public delegate void HeapGoing();

        public static event HeapGoing heapGoing;

        public static void GoToHeapStart()
        {
            heapGoing.Invoke();
        }
        //-------------HeapEvent(Fights,TakeRes)---------------
        public delegate void StartHeapEVENT();

        public static event StartHeapEVENT startHeapEvent;

        public static void HeapEvent()
        {
            startHeapEvent.Invoke();
        }
        //---------------DeleteDeadAnts-------------------------
        public delegate void DeleteDeadUNITS_BornIns_Screen3(int day);

        public static event DeleteDeadUNITS_BornIns_Screen3 DeleteDead_Borning_print;

        public static void DeletingAnts_BornIns_printScreen3(int day)
        {
            DeleteDead_Borning_print.Invoke(day);

        }
    }
```

В методе Main (Листинг 10) назначается первый день, создаются первые 2 колонии. С помощью цикла while и с помощью событий происходит выполнение программы и вывод информации. Для выявления победителя, у каждой колонии подсчитывается общее количество ресурсов и сравнивается с другими.

Листинг 10 – Метод исполнения программы

```
static void Main(string[] args)
        {
            //Начальные значения
            int Days = 1;
            List<Colony> colonies=new List<Colony>()
            {
                Globals.Create_Orange_Colony(),
                Globals.Create_Black_Colony()
            };
            Squirrel squirrel = new Squirrel();
            //основная часть игры
```

```
            while (Days != 14)
            {
                squirrel.SquirrelAttacky(Days);
                Screen_1(Days,squirrel);
                Screen_2();
                DayController.GoToHeapStart();  //распределение по кучам
                Screen_3_HeakStart();
                DayController.HeapEvent();
                DayController.DeletingAnts_BornIns_printScreen3(Days);
                Heaps.HeapDepletion();
                Days++;
                Console.WriteLine("Нажмите ВВОД для перехода на следующий день...");
                Console.ReadKey();
            }

            Console.WriteLine("----------НАСТУПИЛА ЗАСУХА------");
            Colony Winner = WinnerColony(colonies);
            Console.WriteLine($"Колония {Winner.name} выживает после засухи с
{Winner.WinnerPart()} ресурсами!!!");
            Console.WriteLine("THE END!!!!");
            Console.ReadLine();
        }
```

Пример исполнения программы представлен на рис. 1, рис. 2, рис. 3.

```
----------------Экран 1 - Начало хода----------------
День 1 (до засухи осталось 13 дней)
Колония РЫЖИЕ:
---Королева "Шарлотта", личинок: 0
---Ресурсы: в=0, л=0, к=0, р=0
---Популяция 23: р=8 , в=14 , о=1;

Колония ЧЕРНЫЕ:
---Королева "Шарлотта", личинок: 0
---Ресурсы: в=0, л=0, к=0, р=0
---Популяция 21: р=5 , в=15 , о=1;

Куча 1: в=10 , л=17 , к=31 , р=45
Куча 2: в=12 , л=18 , к=0 , р=29
Куча 3: в=17 , л=18 , к=23 , р=0
Куча 4: в=29 , л=0 , к=0 , р=15
Куча 5: в=13 , л=34 , к=0 , р=39
Глобальный эффект: <Белка> ворует половину веточек со случайной кучи(в течении 7 дней)
```

Рис. 1 – Вывод информации в начале дня

```
---------------Экран 2 - Информация по колонии---------------
-------------------------------------------------------
Колония РЫЖИЕ
---Королева Шарлотта: здоровье=17, защита=8 , урон=29

<<<<<<<<<<Рабочие>>>>>>>>>
Тип: Старший
---Параметры: здоровье= 2, защита= 1, может брать 1 ресурс: 'веточка или веточка' за раз
---Количество: 2

Тип: Старший неповторимый
---Параметры: здоровье= 2, защита= 1, может брать 1 ресурс: 'камушек или росинка' за раз
---Модификаторы: полностью неуязвим от всех атак(даже смертельных для неуязвимых, игнорирует
---Количество: 3

<<<<<<<<<<Воины>>>>>>>>>>
Тип: Продвинутый
---Параметры: здоровье=6, защита=2, урон=4, может атаковать 2 цели за раз и наносит 1 укус
---Количество: 1

Тип: Продвинутый мстительный
---Параметры: здоровье=6, защита=2, урон=4, может атаковать 2 цели за раз и наносит 1 укус
---Модификаторы: убивает своего убийцу, даже если он неуязвим
---Количество: 1

<<<<<<<<<<Особое насекомое>>>>>>>>>
Тип: Ленивый обычный агрессивный настойчивый - Шмель
---Параметры: здоровье=25, защита=6, урон=8 наносит 1 укус(а) и атакует 2 за раз
---Модификаторы: всегда наносит укус, даже если был убит,
-----------------------------------------------
```

Рис. 2 – Вывод информации о колонии

```
----------------Экран 3 - Поход---------------
Начало дня:
С колонии РЫЖИЕ отправились: р=3, в=3, о=1 на кучу 1
С колонии РЫЖИЕ отправились: р=2, в=1, о=1 на кучу 2
С колонии РЫЖИЕ отправились: р=4, в=2, о=1 на кучу 3
С колонии РЫЖИЕ отправились: р=2, в=1, о=1 на кучу 4
С колонии РЫЖИЕ отправились: р=3, в=1, о=1 на кучу 5
С колонии ЧЕРНЫЕ отправились: р=4, в=1, о=1 на кучу 1
С колонии ЧЕРНЫЕ отправились: р=4, в=1, о=1 на кучу 2
С колонии ЧЕРНЫЕ отправились: р=5, в=1, о=1 на кучу 3
С колонии ЧЕРНЫЕ отправились: р=2, в=0, о=1 на кучу 4
С колонии ЧЕРНЫЕ отправились: р=0, в=2, о=1 на кучу 5
С колонии РЫЖИЕ вернулись:
---р=9, в=5, о=1
---Потери: р=5, в=3, о=0
---Добыто ресурсов: в=6, л=0, к=2, р=1
---Новые личинки: 4 (еще 3 дней)
С колонии ЧЕРНЫЕ вернулись:
---р=11, в=4, о=1
---Потери: р=4, в=1, о=0
---Добыто ресурсов: в=4, л=7, к=4, р=4
---Новые личинки: 5 (еще 4 дней)
Нажмите ВВОД для перехода на следующий день...
```

Рис.3 – Вывод информации о походе

Реализация дополнительного задания

Реализация дополнительного задания происходит в классе Squirrel, в котором есть поля длительности, дня начала нападения белки, и булевая переменная отвечающая за то, атакует ли в данный момент белка. Функция SquirrelAttacky проверяет булевую переменную атаки. Если она атакует, то выбирается случайная куча и она отнимает половину веточек у этой кучи, если же она не атакует, то с определенной вероятностью она может начать атаковать.

Листинг 13 – Класс Squirrel

```
public class Squirrel
    {
        public int Duration;
        public int DayOfAttack;
        public bool attacking;

        public Squirrel()
        {
            Duration = 7;
            DayOfAttack = 0;
            attacking = false;
        }

        public void SquirrelAttacky(int Day)
        {
            if (!attacking)
            {
                int tmp = Globals._random.Next(0, 3);
                if (tmp == 0)
                {
                    attacking = true;
                    DayOfAttack = Day;
                }
            }
            if (DayOfAttack + Duration == Day && attacking)
            {
                attacking = false;
            }
            if (attacking)
            {
                bool open = true;
                while (open)
                {
                    int tmp = Globals._random.Next(0, Heaps.heaps.Count);
                    if (Heaps.heaps[tmp].IsOpen)
                    {
                        Heaps.heaps[tmp].amountOfRes[0] -=
(Heaps.heaps[tmp].amountOfRes[0]) / 2;
                        open = false;
                    }
                }
            }
        }
    }
```

Список использованных источников

1. Шилдт, Г. C# 4.0: полное руководство / Г. Шилдт; пер. с англ. – М.: ООО «И.Д. Вильямс», 2011 – 1056 с.: ил. – Парал. тит. англ.
2. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Дж. Рихтер; пер. с англ. – СПб.: Питер, 2013. – 896 с.: ил. – (Серия «Мастер-класс»).
3. Хейлсберг, А., Торгерсен М., Вилтамут С., Голд П. Язык программирования C#. Классика Computers science. 4-е изд. / А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. – СПб.: Питер, 2012. – 784 с.: ил.

Приложение

Исходный код программы:

```
using System;
using System.Collections.Generic;
using System.Threading.Channels;

namespace Kursovaia
{
    class Program
    {
        static void Main(string[] args)
        {
            //Начальные значения
            int Days = 1;
            List<Colony> colonies=new List<Colony>()
            {
                Globals.Create_Orange_Colony(),
                Globals.Create_Black_Colony()
            };
            Squirrel squirrel = new Squirrel();
            //основная часть игры
            while (Days != 14)
            {
                squirrel.SquirrelAttacky(Days);
                Screen_1(Days,squirrel);
                Screen_2();
                DayController.GoToHeapStart();  //распределение по кучам
                Screen_3_HeakStart();
                DayController.HeapEvent();
                DayController.DeletingAnts_BornIns_printScreen3(Days);
                Heaps.HeapDepletion();
                Days++;
                Console.WriteLine("Нажмите ВВОД для перехода на следующий день...");
                Console.ReadKey();
            }

            Console.WriteLine("----------НАСТУПИЛА ЗАСУХА------");
            Colony Winner = WinnerColony(colonies);
            Console.WriteLine($"Колония {Winner.name} выживает после засухи с
{Winner.WinnerPart()} ресурсами!!!");
            Console.WriteLine("THE END!!!!");
            Console.ReadLine();
        }
        public static void Screen_1(int day,Squirrel squirrel)
        {
            Console.WriteLine("----------------Экран 1 - Начало хода----------------
");
            Console.WriteLine($"День {day} (до засухи осталось {14-day} дней)");
            DayController.Screen_1();
            foreach (Heap heap in Heaps.heaps)
            {
                if (heap.IsOpen)
                {
                    Console.WriteLine(
                        $"Куча {heap.HeapNum}: в={heap.amountOfRes[0]} ,
л={heap.amountOfRes[1]} , к={heap.amountOfRes[2]} , р={heap.amountOfRes[3]}");
                }
                else
                {
                    Console.WriteLine(
                        $"Куча {heap.HeapNum}: истощена");
```

```
                }
            }
            if (squirrel.attacking)
            {
                Console.WriteLine($"Глобальный эффект: <Белка> ворует половину
веточек со случайной кучи(в течении {squirrel.DayOfAttack+squirrel.Duration-day}
дней)");
            }
        }

        public static void Screen_2()
        {
            Console.WriteLine("---------------Экран 2 - Информация по колонии--------
--------");
            DayController.Screen_2();
        }
        public static void Screen_3_HeakStart()
        {
            Console.WriteLine("----------------Экран 3 - Поход---------------");
            Console.WriteLine("Начало дня: ");
            DayController.Screen_3_Start();
        }
        public static Colony WinnerColony(List<Colony> colonies)
        {
            foreach (Colony friendlyColony in colonies[0].FriendlyColonies)
            {
                colonies.Add(friendlyColony);
            }
            foreach (Colony friendlyColony in colonies[1].FriendlyColonies)
            {
                colonies.Add(friendlyColony);
            }
            int tmp = 0;
            Colony winner = new Colony();
            foreach (Colony colony in colonies)
            {
                if (colony.WinnerPart() > tmp)
                {
                    winner = colony;
                    tmp = colony.WinnerPart();
                }
            }
            return winner;
        }
    }

public class Ant:ITakeDamage
    {
        public int hp;
        public int def;
        public int damage;
        public string name;
        public bool IsAlive;
        public Colony _colony;
        public Ant(string name, int hp, int def, int damage,Colony _colony)
        {
            this.hp = hp;
            this.def = def;
            this.damage = damage;
            this.name = name;
            IsAlive = true;
            this._colony = _colony;
        }
        public virtual void TakeDamage(int dam,bool ignoreDef)
```

```
        {
            IsAlive = false;
        }
    }
    //---------------Список типов муравьев----------------------
    public enum WorkerType
    {
        USUAL,
        OLDER,
        OLDER_UNIQUE,
        LEGENDARY,
        LEGENDARY_RUNNER
    }
    public enum WariourType
    {
        USUAL,
        ADVANCED,
        ADVANCED_REVENGEFULL,
        OLDER,
        ELITE,
        LEGENDARY_FAT
    }

public interface IAttack:ITakeDamage,ICanGoToHeap
    {
        int Attack();
        int TargetsOnHeap();
    }
    public interface ITakeDamage
    {
        void TakeDamage(int dam,bool ignoreDef);
    }

public class Queen:Ant
    {
        public int[] Borning;
        public int[] Queens;
        public List<WariourType> wariorsType;
        public List<WorkerType> workersType;
        public int DayOfEggBirth;
        public int eggs;
        public int BorningTime;
        public int BornedQueens;
        public Queen(string name, int hp, int def, int damage,Colony _colony, int[]
borning, int[] queens,List<WariourType> wariorsType,List<WorkerType> workersType) :
base(name,hp, def, damage,_colony)
        {
            this.wariorsType = wariorsType;
            this.workersType = workersType;
            Borning = borning;
            Queens = queens;
            eggs = 0;
            BornedQueens = 0;
        }
        //-------------------Рождение муравьев------------------
        public void BornInsects(int Day)
        {
            if (eggs == 0)
            {
                eggs = Globals._random.Next(2, 10);
                DayOfEggBirth = Day;
                BorningTime = Globals._random.Next(Borning[0], Borning[1]+1);
                Console.WriteLine($"---Новые личинки: {eggs} (еще {BorningTime-
Day+DayOfEggBirth} дней)");
```

```
            }
            else if (Day == (DayOfEggBirth + BorningTime))
            {
                int newWorkers = 0;
                int newWariours = 0;
                int newQueen = 0;
                int CanBornQueens = 3;
                while (eggs != 0)
                {
                    if (BornedQueens >= Queens[1])
                    {
                        CanBornQueens = 2;
                    }
                    switch (Globals._random.Next(0, CanBornQueens))
                    {
                        case 0:
                        {
                            WariourType tmp = wariousType[Globals._random.Next(0,
wariousType.Count)];
                            switch (tmp)
                            {
                                case WariourType.USUAL:
_colony.units_Wariour.Add(Globals.Create_Usual_Wariour(_colony));
                                    break;
                                case WariourType.ELITE:
_colony.units_Wariour.Add(Globals.Create_Elite_Wariour(_colony));
                                    break;
                                case WariourType.OLDER:
_colony.units_Wariour.Add(Globals.Create_Older_Wariour(_colony));
                                    break;
                                case WariourType.ADVANCED:
_colony.units_Wariour.Add(Globals.Create_Advanced_Wariour(_colony));
                                    break;
                                case WariourType.LEGENDARY_FAT:
_colony.units_Wariour.Add(Globals.Create_LegendaryFat_Wariour(_colony));
                                    break;
                                case WariourType.ADVANCED_REVENGEFULL:
_colony.units_Wariour.Add(Globals.Create_AdvancedRevengefull_Wariour(_colony));
                                    break;
                            }

                            newWariours++;
                            break;
                        }
                        case 1:
                        {
                            WorkerType tmp = workersType[Globals._random.Next(0,
workersType.Count)];
                            switch (tmp)
                            {
                                case WorkerType.OLDER:
_colony.units_Worker.Add(Globals.Create_Older_Worker(_colony));
                                    break;
                                case WorkerType.USUAL:
_colony.units_Worker.Add(Globals.Create_Usual_Worker(_colony));
                                    break;
                                case WorkerType.LEGENDARY:
_colony.units_Worker.Add(Globals.Create_Legendary_Worker(_colony));
                                    break;
                                case WorkerType.OLDER_UNIQUE:
_colony.units_Worker.Add(Globals.Create_OlderUnique_Worker(_colony));
                                    break;
```

```
                                case WorkerType.LEGENDARY_RUNNER:
_colony.units_Worker.Add(Globals.Create_LegendaryRunner_Worker(_colony));
                                    break;
                            }
                            newWorkers++;
                            break;
                        }
                        case 2:
                        {
                            BornedQueens++;
                            int tmp = Globals._random.Next(0, 2);
                            if ((_colony.FriendlyColonies.Count <
Queens[1])&&(tmp==0))
                            {
                                Globals.FriendlyColony_Creation(_colony);
                            }
                            newQueen++;
                            break;
                        }
                    }
                    eggs--;
                }
                eggs = Globals._random.Next(2, 10);
                DayOfEggBirth = Day;
                BorningTime = Globals._random.Next(Borning[0], Borning[1]+1);
                Console.WriteLine($"---Выросли: р={newWorkers}, в={newWariours},
к={newQueen} \n" +
                                  $"---Новые личинки: {eggs} (еще {BorningTime-
Day+DayOfEggBirth} дней)");
            }
            else
            {
                Console.WriteLine($"---Личинки еще растут({BorningTime-
Day+DayOfEggBirth})");
            }
        }
    }

public class Worker:Ant,ICanGoToHeap
    {
        //-------------Значения---------------
        public Resourses TakableResourses;
        public Resourses Backpack;
        public bool TakeResOr_And;
        public Heap _heap;
        public WorkerType _Type;
        public List<Modificator> Modificators;
        //------------------Конструкторы----------------------

        public Worker(string name, int hp, int def, int damage,Colony
_colony,WorkerType _Type,Resourses takableResources,bool takeResOrAnd) :
base(name,hp, def, damage,_colony)
        {
            this.TakableResourses = takableResources;
            this.Backpack = new Resourses(0, 0, 0, 0);
            this._Type = _Type;
            this.TakeResOr_And = takeResOrAnd;     //если true то and, если false то
or!!!
            _heap = null;
            Modificators = new List<Modificator>();
            DayController.heapGoing+=GoToHeap;
            DayController.DeleteDead_Borning_print+=DeleteDeadBorningPrint;
        }
```

```
        public Worker(string name, int hp, int def, int damage,Colony
_colony,WorkerType _Type,Resourses takableResources,bool
takeResOrAnd,List<Modificator> Modificators) : base(name,hp, def, damage,_colony)
        {
            this.TakableResourses = takableResources;
            this.Backpack = new Resourses(0, 0, 0, 0);
            this._Type = _Type;
            this.TakeResOr_And = takeResOrAnd;     //если true то and, если false то
or!!!
            _heap = null;
            this.Modificators = Modificators;
            DayController.heapGoing+=GoToHeap;
            DayController.DeleteDead_Borning_print+=DeleteDeadBorningPrint;
        }
        //-----------------Получение урона----------------------
        public override void TakeDamage(int dam,bool ignoreDef)
        {
            if (!(this.Modificators.Contains(Modificator.Immortal)))
            {
                IsAlive = false;
            }

        }
        //----------------------Назначение кучи-------------------
        public void GoToHeap()
        {
            bool open = true;
            while (open)
            {
                _heap = Heaps.heaps[Globals._random.Next(0, Heaps.heaps.Count)];
                if (_heap.IsOpen)
                {
                    _heap.ComeIn(this);
                    open = false;
                }
            }
        }
        //-----------------Удаление при смерти----------------------
        public void DeleteDeadBorningPrint(int none)
        {
            if (IsAlive == false)
            {
                DayController.heapGoing -= GoToHeap;
                DayController.DeleteDead_Borning_print -= DeleteDeadBorningPrint;
            }
        }
        //-----------------Получение ресурсов--------------------
         public void TryTakeResources()
        {
            if (TakeResOr_And)
            {
                for (int i = 0; i < 4; i++)
                {
                    if ((TakableResourses.amountOfRes[i] >
0)&&(Modificators.Contains(Modificator.FindResAnywhere)))
                    {
                        if (_heap.amountOfRes[i] <= 0)
                        {
                            Backpack.amountOfRes[i] +=
TakableResourses.amountOfRes[i];
                        }
                        else
                        {
```

```
                            _heap.amountOfRes[i] -= TakableResourses.amountOfRes[i];
                            Backpack.amountOfRes[i] +=
TakableResourses.amountOfRes[i];
                        }
                    }
                    else if ((TakableResourses.amountOfRes[i] >
0)&&(_heap.amountOfRes[i]>0))
                    {
                        _heap.amountOfRes[i] -= TakableResourses.amountOfRes[i];
                        Backpack.amountOfRes[i] += TakableResourses.amountOfRes[i];
                    }
                }
            }
            else
            {
                List<int> amountTypes = new List<int>();

                for (int i = 0; i < 4; i++)
                {
                    if
((Modificators.Contains(Modificator.FindResAnywhere))&&(TakableResourses.amountOfRes[
i] > 0))
                    {
                        amountTypes.Add(i);
                    }
                    else if ((TakableResourses.amountOfRes[i] >
0)&&(_heap.amountOfRes[i]>0))
                    {
                        amountTypes.Add(i);
                    }
                }

                if (amountTypes.Count != 0)
                {
                    int tmp = Globals._random.Next(0, amountTypes.Count);
                    if (_heap.amountOfRes[amountTypes[tmp]] <= 0)
                    {
                        Backpack.amountOfRes[amountTypes[tmp]] +=
TakableResourses.amountOfRes[amountTypes[tmp]];
                    }
                    else
                    {
                        _heap.amountOfRes[amountTypes[tmp]] -=
TakableResourses.amountOfRes[amountTypes[tmp]];
                        Backpack.amountOfRes[amountTypes[tmp]] +=
TakableResourses.amountOfRes[amountTypes[tmp]];
                    }
                }

            }
        }
         //----------------------Функции для интерфейсов-------------
        public List<Modificator> ModificatorsReturn()
        {
            return Modificators;
        }

        public void AddModificatorDeath(Modificator mods)
        {
            this.Modificators.Add(mods);
        }

        public Colony MyColony()
```

```
        {
            return _colony;
        }

        public bool IsDead()
        {
            return IsAlive;
        }
        public void Death()
        {
            IsAlive = false;
        }
        //-------------Вывод информации-------------------
        public void tellAboutYourself()
        {
            switch (_Type)
            {
                case WorkerType.USUAL:
                {
                    Console.WriteLine(
                        $"Тип: Обычный \n"+
                        $"---Параметры: здоровье= 1, защита= 0, может брать 1 ресурс:
'веточка' за раз");
                    break;
                }
                case WorkerType.OLDER:
                {
                    Console.WriteLine(
                        $"Тип: Старший \n"+
                        $"---Параметры: здоровье= 2, защита= 1, может брать 1 ресурс:
'веточка или веточка' за раз");
                    break;
                }
                case WorkerType.OLDER_UNIQUE:
                {
                    Console.WriteLine(
                        $"Тип: Старший неповторимый \n"+
                        $"---Параметры: здоровье= 2, защита= 1, может брать 1 ресурс:
'камушек или росинка' за раз \n" +
                        $"---Модификаторы: полностью неуязвим от всех атак(даже
смертельных для неуязвимых, игнорирует все модификаторы врагов, всегда находит нужный
ресурс в куче(даже если его там нет) ");
                    break;
                }
                case WorkerType.LEGENDARY:
                {
                    Console.WriteLine(
                        $"Тип: Легендарный \n"+
                        $"---Параметры: здоровье= 10, защита= 6, может брать 3
ресурса: 'листик и камушек и росинка' за раз ");
                    break;
                }
                case WorkerType.LEGENDARY_RUNNER:
                {
                    Console.WriteLine(
                        $"Тип: Легендарный спринтер \n"+
                        $"---Параметры: здоровье= 10, защита= 6, может брать 3
ресурса: 'листик и камушек и росинка' за раз \n" +
                        $"---Модификаторы: не может быть атакован первым ");
                    break;
                }
            }
            Console.WriteLine($"---Королева: {_colony.Queen.name}");
```

```
        }
    }

public class Wariour:Ant,IAttack,ICanGoToHeap
    {
        //------------Значения---------------
        public WariourType _Type;
        public int targets;
        public int bites;
        public Heap _heap;
        public List<Modificator> Modificators;
        public int targetsInHeak;
        //-----------Конструкторы----------------
        public Wariour(string name, int hp, int def, int damage,Colony
_colony,WariourType _type, int targets, int bites) : base(name,hp, def,
damage,_colony)
        {
            this._Type = _type;
            this.targets = targets;
            this.bites = bites;
            _heap = null;
            Modificators = new List<Modificator>();
            targetsInHeak = targets;
            DayController.heapGoing += GoToHeap;
            DayController.DeleteDead_Borning_print+=DeleteDeadBorningPrint;
        }
        public Wariour(string name, int hp, int def, int damage,Colony
_colony,WariourType _type, int targets, int bites,List<Modificator> Modificators) :
base(name,hp, def, damage,_colony)
        {
            this.Modificators = Modificators;
            this._Type = _type;
            this.targets = targets;
            this.bites = bites;
            _heap = null;
            targetsInHeak = targets;
            if (Modificators.Contains(Modificator.DoubleHPDEF))
            {
                this.hp *= 2;
                this.def *= 2;
            }
            DayController.heapGoing += GoToHeap;
            DayController.DeleteDead_Borning_print+=DeleteDeadBorningPrint;
        }
        //--------------Получение урона--------------
        public override void TakeDamage(int dam, bool ignoreDef)
        {
            if (ignoreDef)
            {
                hp -= dam;
                if (hp <= 0)
                {
                    IsAlive = false;
                }
            }
            else
            {
                if (dam < def)
                {
                    hp -= 1;
                }
                else if (dam == def)
                {
                    hp -= dam / 2;
```

```
            }
            else
            {
                hp -= dam - def / 2;
            }

            if (hp <= 0)
            {
                IsAlive = false;
                DayController.heapGoing -= GoToHeap;
            }
        }
    }
    //-----------------Назначение куче-----------------
    public void GoToHeap()
    {
        bool open = true;
        while (open)
        {
            _heap = Heaps.heaps[Globals._random.Next(0, Heaps.heaps.Count)];
            if (_heap.IsOpen)
            {
                _heap.ComeIn(this);
                open = false;
            }
        }
    }
    //---------------Удаление муравья если мертв, либо возвращение целей---------
---
    public void DeleteDeadBorningPrint(int none)
    {
        targetsInHeak = targets;
        if (IsAlive == false)
        {
            DayController.heapGoing -= GoToHeap;
            DayController.DeleteDead_Borning_print -= DeleteDeadBorningPrint;
        }
    }
    //------------------Функции для ИНТЕРФЕЙСОВ-------------------
    public int TargetsOnHeap()
    {
        return targetsInHeak;
    }

    public void Death()
    {
        IsAlive = false;
    }

    public void AddModificatorDeath(Modificator mods)
    {
        this.Modificators.Add(mods);
    }
    public List<Modificator> ModificatorsReturn()
    {
        return Modificators;
    }

    public int Attack()
    {
        targetsInHeak -= 1;
        return (damage*bites);
    }
```

```
        public bool IsDead()
        {
            return IsAlive;
        }
        public Colony MyColony()
        {
            return _colony;
        }
        //----------------Вывод информации--------------------
        public void tellAboutYourself()
        {
            switch (_Type)
            {
               case WariourType.USUAL:
               {
                   Console.WriteLine("Тип: Обычный");
                   break;
               }
               case WariourType.ADVANCED:
               {
                   Console.WriteLine("Тип: Продвинутый");
                   break;
               }
               case WariourType.ADVANCED_REVENGEFULL:
               {
                   Console.WriteLine("Тип: Продвинутый мстительный");
                   break;
               }
               case WariourType.OLDER:
               {
                   Console.WriteLine("Тип: Старший");
                   break;
               }
               case WariourType.ELITE:
               {
                   Console.WriteLine("Тип: Элитный");
                   break;
               }
               case WariourType.LEGENDARY_FAT:
               {
                   Console.WriteLine("Тип: Легендарный толстый");
                   break;
               }
            }
            Console.WriteLine($"---Параметры: здоровье={hp}, защита={def},
урон={damage}, может атаковать {targets} цель за раз и наносит {bites} укус");
            Console.Write($"---Модификаторы: ");
            foreach (string printModificator in Globals.print_Modificators(this))
            {
                Console.Write(printModificator+", ");
            }

            Console.WriteLine($"---Королева: {_colony.Queen.name}");
        }
    }

public class SpecialInsect:Ant,ICanGoToHeap,IAttack
    {
        //-----------------Значения-------------
        public List<Modificator> Modificators;
        public Heap _heap;
        public int targets;
        public int bites;
        public int targetsInHeak;
```

```
        //----------------Конструкторы----------------
        public SpecialInsect(string name, int hp, int def, int damage, Colony
_colony,  int targets, int bites,List<Modificator> modificators) : base(name, hp,
def, damage, _colony)
        {
            Modificators = modificators;
            _heap = null;
            this.targets = targets;
            this.bites = bites;
            targetsInHeak = targets;
            DayController.heapGoing += GoToHeap;
            DayController.DeleteDead_Borning_print+=DeleteDeadBorningPrint;
        }
        //-------------------Получение урона--------------------
        public override void TakeDamage(int dam,bool ignoreDef)
        {
            if (ignoreDef)
            {
                hp -= dam;
                if (hp <= 0)
                {
                    IsAlive = false;
                }
            }
            else
            {
                if (dam < def)
                {
                    hp -= 1;
                }
                else if (dam == def)
                {
                    hp -= dam / 2;
                }
                else
                {
                    hp -= dam - def / 2;
                }

                if (hp <= 0)
                {
                    IsAlive = false;
                    DayController.heapGoing -= GoToHeap;
                }
            }
        }
        //---------------------Назначение кучи----------------------
        public void GoToHeap()
        {
            bool open = true;
            while (open)
            {
                _heap = Heaps.heaps[Globals._random.Next(0, Heaps.heaps.Count)];
                if (_heap.IsOpen)
                {
                    _heap.ComeIn(this);
                    open = false;
                }
            }
        }
        //------------------Удаление мертвых------------------
        public void DeleteDeadBorningPrint(int none)
        {
```

```
            targetsInHeak = targets;
            if (IsAlive == false)
            {
                DayController.heapGoing -= GoToHeap;
                DayController.DeleteDead_Borning_print -= DeleteDeadBorningPrint;
            }
        }
        //---------------Функции для ИНТЕРФЕЙСОВ------------------
        public int TargetsOnHeap()
        {
            return targetsInHeak;
        }
        public void Death()
        {
            IsAlive = false;
        }
        public List<Modificator> ModificatorsReturn()
        {
            return Modificators;
        }
        public int Attack()
        {
            targetsInHeak -= 1;
            return (damage*bites);
        }
        public bool IsDead()
        {
            return IsAlive;
        }
        public void AddModificatorDeath(Modificator mods)
        {
            this.Modificators.Add(mods);
        }
        public Colony MyColony()
        {
            return _colony;
        }
        //-------------------------ВЫВОД ИНФОРМАЦИИ---------------------
        public void tellAboutYou()
        {

            Console.Write($"Тип: {name} \n" +
                          $"---Параметры: здоровье=25, защита={def},
урон={damage} наносит {bites} укус(а) и атакует {targets} за раз \n" +
                          $"---Модификаторы: ");
            foreach (string modificator in Globals.print_Modificators(this))
            {
                Console.Write(modificator+", ");
            }

            Console.WriteLine();
        }
    }

public class Colony
    {
        //---------------Юниты--------------
        public Queen Queen;
        public List<Wariour> units_Wariour;
        public List<Worker> units_Worker;
        public List<SpecialInsect> SpecialInsect;
        //--------------Значения колонии-------------
        public string name;
        public ColonyTypes ColonyType;
```

```
        public Resourses ColonyRes;
        public List<Colony> FriendlyColonies;
        //--------------------Конструктор-------------------
        public Colony()
        {
            this.units_Wariour = new List<Wariour>();
            this.units_Worker = new List<Worker>();
            this.SpecialInsect = new List<SpecialInsect>();
            this.ColonyRes = new Resourses(0, 0, 0, 0);
            DayController.screen_1 += print_Screen_1;
            DayController.screen_2 += print_Screen_2;
            DayController.screen_3_HeakStart += Screen_3_HeakStart;
            DayController.DeleteDead_Borning_print += DelDeadPrintAnts;
            DayController.DeleteDead_Borning_print += GetResourses_BornInsects;
            FriendlyColonies = new List<Colony>();
        }
        //-------------------------Удаление мертвых муравьев-------------------
        public void DelDeadPrintAnts(int day)
        {
            int deadWariours = 0;
            int deadWorkers = 0;
            int deadSpecIns = 0;
            for (int i = 0; i < units_Wariour.Count; i++)
            {
                if (!(units_Wariour[i].IsAlive))
                {
                    units_Wariour.RemoveAt(i);
                    deadWariours++;
                }
            }

            for (int i = 0; i < units_Worker.Count; i++)
            {
                if (!(units_Worker[i].IsAlive))
                {
                    units_Worker.RemoveAt(i);
                    deadWorkers++;
                }
            }
            for (int i = 0; i < SpecialInsect.Count; i++)
            {
                if (SpecialInsect[i].IsAlive == false)
                {
                    SpecialInsect.RemoveAt(i);
                    deadSpecIns++;
                }
            }

            Console.WriteLine($"С колонии {name} вернулись: \n" +
                              $"---р={units_Worker.Count}, в={units_Wariour.Count},
o={SpecialInsect.Count} \n" +
                              $"---Потери: р={deadWorkers}, в={deadWariours},
o={deadSpecIns}");
        }
        //----------------------Получение ресурсов от рабочих-----------------------
        public void GetResourses_BornInsects(int day)
        {
            int[] gotRes = new int[4] {0, 0, 0, 0};
            for (int i = 0; i < units_Worker.Count; i++)
            {
                for (int k = 0; k < 4; k++)
                {
```

```
                    this.ColonyRes.amountOfRes[k] +=
units_Worker[i].Backpack.amountOfRes[k];
                    gotRes[k]+=units_Worker[i].Backpack.amountOfRes[k];
                }

                units_Worker[i].Backpack.amountOfRes = new int[4] {0, 0, 0, 0};
            }

            Console.WriteLine($"---Добыто ресурсов: в={gotRes[0]}, л={gotRes[1]},
к={gotRes[2]}, р={gotRes[3]}");
            Queen.BornInsects(day);
        }
        //----------------------Вывод количества всех ресурсов-------------------
        public int WinnerPart()
        {
            int amount = 0;
            foreach (int i in ColonyRes.amountOfRes)
            {
                amount += i;
            }

            return amount;
        }
        //---------------------------ВЫВОД ИНФОРМАЦИИ-------------------------------
        public void print_Screen_1()
        {
            Console.WriteLine($"Колония {this.name}: \n" +
                              $"---Королева \"{Queen.name}\", личинок: {Queen.eggs}
\n" +
                              $"---Ресурсы: в={ColonyRes.amountOfRes[0]},
л={ColonyRes.amountOfRes[1]}, к={ColonyRes.amountOfRes[2]},
р={ColonyRes.amountOfRes[3]} \n" +
                              $"---Популяция
{units_Wariour.Count+units_Worker.Count+SpecialInsect.Count}: р={units_Wariour.Count}
, в={units_Worker.Count} , о={SpecialInsect.Count};");
            Console.WriteLine();
        }

        public void print_Screen_2()
        {
            Console.WriteLine("-----------------------------------------------------
--");
            Console.WriteLine($"Колония {name} \n" +
                              $"---Королева {Queen.name}: здоровье={Queen.hp},
защита={Queen.def} , урон={Queen.damage}");
            Console.WriteLine();
            if (units_Worker.Count != 0)
            {
                Console.WriteLine("<<<<<<<<<<Рабочие>>>>>>>>>>");
            }

            foreach (WorkerType type in Queen.workersType)
            {
                int count = 0;

                foreach (Worker worker in units_Worker)
                {
                    if (worker._Type == type)
                    {
                        count++;
                    }
                }
                if (count != 0)
```

```
            {
                Globals.print_About_Workers(type);
                Console.WriteLine($"---Количество: {count}");
                Console.WriteLine();
            }
        }

        if (units_Wariour.Count != 0)
        {
            Console.WriteLine("<<<<<<<<<<Воины>>>>>>>>>>");
        }

        foreach (WariourType type in Queen.wariorsType)
        {
            int count = 0;
            foreach (Wariour wariour in units_Wariour)
            {
                if (wariour._Type == type)
                {
                    count++;
                }
            }

            if (count != 0)
            {
                Globals.print_About_Wariours(type);
                Console.WriteLine($"---Количество: {count}");
                Console.WriteLine();
            }
        }

        if (SpecialInsect.Count != 0)
        {
            Console.WriteLine("<<<<<<<<<<<Особое насекомое>>>>>>>>>");
        }

        foreach (SpecialInsect specialInsect in SpecialInsect)
        {
            specialInsect.tellAboutYou();
        }
        Console.WriteLine("-----------------------------------------------");
    }

    public void Screen_3_HeakStart()
    {
        //Workers
        foreach (Heap heap in Heaps.heaps)
        {
            int workersCount = 0;
            foreach (Worker worker in units_Worker)
            {
                if (worker._heap == heap)
                {
                    workersCount++;
                }
            }
            int wariorsCount = 0;
            foreach (Wariour wariour in units_Wariour)
            {
                if (wariour._heap == heap)
                {
                    wariorsCount++;
                }
```

```
                }

                int specInsCount = 0;
                foreach (SpecialInsect specialInsect in SpecialInsect)
                {
                    specInsCount++;
                }
                Console.WriteLine($"С колонии {name} отправились: р={workersCount},
в={wariorsCount}, о={specInsCount} на кучу {heap.HeapNum}");
            }
        }
    }

    public enum ColonyTypes
    {
        Orange,
        Black
    }

    public enum Modificator
    {
        //Worker MOds
        Immortal,    //неуязвимость +
        ModIgnoting,   //Игнорирование модификаторов – невозможно представить
        FindResAnywhere,// всегда находит ресурс в куче +
        CantBeAttackedFirst, //не может быть атакован первым +

        //Wariours Mods
        KillKiller,  //убивает своего убийцу +
        Provocation,   //принимает все атаки на себя +
        DoubleHPDEF,   //здоровье и защита увеличены в 2 раза +

        //SpecialInsect Mods
        BytesAnyway, //Наносит укус даже если был убит +
        CantBeAttacked, //Не может быть атакован +
        IgnoreDef, //Игнорирует защиту +
        CanDamageImmortals  //Может наность урон неуязвимым врагам – невозможно
представит
    }

public class Resourses
    {
        public Resources[] typeOfRes;       //порядок: веточка -> листик -> камень ->
капля
        public int[] amountOfRes;
        public Resourses(int branch,int leaf,int stone,int drop)
        {
            this.typeOfRes = new Resources[4] {Resources.Branch, Resources.Leaf,
Resources.Stone, Resources.Drop};
            this.amountOfRes = new int[4] {branch, leaf, stone, drop};
        }
    }
    public enum Resources
    {
        Branch,
        Stone,
        Drop,
        Leaf
    }

public class Heap: Resourses
    {
        public List<ICanGoToHeap> units;
        public int HeapNum;
```

```
        public bool IsOpen;
        public Heap(int branch, int leaf, int stone, int drop,int heapNum) :
base(branch, leaf, stone, drop)
        {
            this.HeapNum = heapNum;
            units = new List<ICanGoToHeap>();
            IsOpen = true;
            DayController.startHeapEvent += Hike;
        }
        //----------------------Юниты на куче-------------------
        public void ComeIn(ICanGoToHeap unit)
        {
            units.Add(unit);
        }
        //-----------------------
        public void DeleteHeap()
        {
            if (IsOpen)
            {
                IsOpen = false;
                DayController.startHeapEvent -= Hike;
            }
        }
        //-------------------------ПОХОД----------------------------
        public void Hike()
        {
            randomQueue();
            for (int i = 0; i < units.Count; i++)
            {
                if (units[i] is Worker)
                {
                    Worker worker = (Worker) units[i];
                    worker.TryTakeResources();
                }
            }
            int amountOfFights = 0;
            for (int i = 0; i < units.Count; i++)
            {
                List<int> IndexOfDeadInsects = new List<int>();
                if ((units[i] is IAttack))
                {
                    IAttack wariour = (IAttack) units[i];
                    bool flagOfNoTarget = false;
                    int targets = wariour.TargetsOnHeap();
                    while (targets > 0)
                    {
                        List<int> PossibleTargetsIndex = ExistenceOfEnemi(units,
wariour, amountOfFights);
                        if (PossibleTargetsIndex.Count == 0)
                        {
                            flagOfNoTarget = true;
                            break;
                        }
                        bool flag = true;
                        foreach (int target in PossibleTargetsIndex)
                        {
                            if
(units[target].ModificatorsReturn().Contains(Modificator.Provocation))
                            {
                                flag = false;
                                Battle.Fight(wariour, units[target]);
                                amountOfFights += 1;
```

```
                                if
((!units[target].IsDead())&&units[target].ModificatorsReturn().Contains(Modificator.K
illKiller))
                                {
                                    wariour.Death();
                                }
                                if (!wariour.IsDead())
                                {
                                    if
(wariour.ModificatorsReturn().Contains(Modificator.KillKiller))
                                    {
                                        units[target].Death();
                                    }

units[i].AddModificatorDeath(Modificator.CantBeAttacked);
                                    if
((!units[i].ModificatorsReturn().Contains(Modificator.BytesAnyway))||wariour.TargetsO
nHeap()==0)
                                    {
                                        IndexOfDeadInsects.Add(i);
                                        targets = 1;
                                    }
                                }
                                if (!units[target].IsDead())
                                {

units[target].AddModificatorDeath(Modificator.CantBeAttacked);
                                    if
((!units[target].ModificatorsReturn().Contains(Modificator.BytesAnyway)))
                                    {
                                        IndexOfDeadInsects.Add(target);
                                    }
                                    else
                                    {
                                        if (((IAttack) units[target]).TargetsOnHeap()
== 0)
                                        {
                                            IndexOfDeadInsects.Add(target);
                                        }
                                    }
                                }
                                break;
                            }
                        }
                        if (flag)
                        {
                            int targetInd =
Globals._random.Next(0,PossibleTargetsIndex.Count);
                            int target = PossibleTargetsIndex[targetInd];
                            Battle.Fight(wariour, units[target]);
                            amountOfFights++;
                            if
((!units[target].IsDead())&&units[target].ModificatorsReturn().Contains(Modificator.K
illKiller))
                            {
                                wariour.Death();
                            }
                            if (!wariour.IsDead())
                            {
                                if
(wariour.ModificatorsReturn().Contains(Modificator.KillKiller))
                                {
                                    units[target].Death();
```

```
                                }

units[i].AddModificatorDeath(Modificator.CantBeAttacked);
                                if
((!units[i].ModificatorsReturn().Contains(Modificator.BytesAnyway))||wariour.TargetsO
nHeap()==0)
                                {
                                    IndexOfDeadInsects.Add(i);
                                    targets = 1;
                                }
                            }
                            if (!units[target].IsDead())
                            {

units[target].AddModificatorDeath(Modificator.CantBeAttacked);
                                if
((!units[target].ModificatorsReturn().Contains(Modificator.BytesAnyway)))
                                {
                                    IndexOfDeadInsects.Add(target);
                                }
                                else
                                {
                                    if (((IAttack) units[target]).TargetsOnHeap() ==
0)
                                    {
                                        IndexOfDeadInsects.Add(target);
                                    }
                                }
                            }

                        }
                        targets--;
                    }
                    if (flagOfNoTarget)
                    {
                        continue;
                    }
                }
                IndexOfDeadInsects.Sort();
                for (int ind = IndexOfDeadInsects.Count - 1; ind >= 0; ind--)
                {
                    units.RemoveAt(IndexOfDeadInsects[ind]);
                    if (IndexOfDeadInsects[ind] < i)
                    {
                        i--;
                    }
                }
            }
            units = new List<ICanGoToHeap>();
        }
        //---------------------Возвращение возможных врагов для определенного юнита--
----------------------
        public List<int> ExistenceOfEnemi(List<ICanGoToHeap> units,ICanGoToHeap
wariour,int amountOfFights)
        {
            List<int> Targets = new List<int>();
            for (int i = 0; i < units.Count; i++)
            {
                bool IsAgresive = true;
                foreach (Colony friendlyColony in
wariour.MyColony().FriendlyColonies)
                {
                    if (friendlyColony == units[i].MyColony())
```

```
                    {
                        IsAgresive = false;
                    }
                }
                if ((units[i].MyColony() != wariour.MyColony()) &&

(!units[i].ModificatorsReturn().Contains(Modificator.CantBeAttacked))&&

((!units[i].ModificatorsReturn().Contains(Modificator.CantBeAttackedFirst))||(units[i
].ModificatorsReturn().Contains(Modificator.CantBeAttackedFirst)&&amountOfFights!=0))
)
                {
                    Targets.Add(i);
                }
            }
            return Targets;
        }

        public void randomQueue()
        {
            for (int i = units.Count - 1; i >= 1; i--)
            {
                int j = Globals._random.Next(i + 1);
                ICanGoToHeap temp = units[j];
                units[j] = units[i];
                units[i] = temp;
            }
        }
    }

public static class Heaps
    {
        public static List<Heap> heaps=Globals.Create_Heaps();
        public static void HeapDepletion()
        {
            for (int i = 0; i < heaps.Count;i++)
            {
                if ((heaps[i].amountOfRes[0] == 0) && (heaps[i].amountOfRes[1] == 0)
&&
                    (heaps[i].amountOfRes[2] == 0) && (heaps[i].amountOfRes[3] == 0))
                {
                    heaps[i].DeleteHeap();
                }
            }
        }
    }

public interface ICanGoToHeap:ITakeDamage
    {
        Colony MyColony();
        bool IsDead();
        List<Modificator> ModificatorsReturn();
        void AddModificatorDeath(Modificator mods);
        void Death();
    }

public static class Battle
    {
        public static void Fight(IAttack fir,ICanGoToHeap sec)
        {

            if (fir.ModificatorsReturn().Contains(Modificator.IgnoreDef))
            {
                IAttack temp = (IAttack) fir;
```

```
                sec.TakeDamage(temp.Attack(),true);
            }
            else
            {
                IAttack temp = (IAttack) fir;
                sec.TakeDamage(temp.Attack(),false);
            }

            if ((sec is
IAttack)&&(!fir.ModificatorsReturn().Contains(Modificator.CantBeAttacked)))
            {
                if (sec.ModificatorsReturn().Contains(Modificator.IgnoreDef))
                {
                    IAttack temp = (IAttack) sec;
                    fir.TakeDamage(temp.Attack(), true);
                }
                else
                {
                    IAttack temp = (IAttack) sec;
                    fir.TakeDamage(temp.Attack(), false);
                }
            }


        }

    }

public static class Globals
    {
        public static Random _random = new Random();
        //
        //   WORKERS CREATION
        //
        public static Worker Create_Usual_Worker(Colony colony)
        {
            return new Worker("Обычный", 1, 0, 0,colony, WorkerType.USUAL,
                new Resourses(1,0,0,0),true);
        }

        public static Worker Create_Older_Worker(Colony colony)
        {
            return new Worker("Старший", 2, 1, 0,colony, WorkerType.OLDER, new
Resourses(1,0,0,0),true);

        }

        public static Worker Create_OlderUnique_Worker(Colony colony)
        {
            return new Worker("Старший неповторимый", 2, 1,
0,colony,WorkerType.OLDER_UNIQUE,new Resourses(0,0,1,1),false,new List<Modificator>()
                {
                 Modificator.Immortal,
              Modificator.ModIgnoting,Modificator.FindResAnywhere});
        }

        public static Worker Create_Legendary_Worker(Colony colony)
        {
            return new Worker("Легендарный", 10, 6, 0,colony,WorkerType.LEGENDARY,new
Resourses(0,1,1,1),true);
        }

        public static Worker Create_LegendaryRunner_Worker(Colony colony)
        {
```

```
            return new Worker("Легендарный спринтер", 10, 6,
0,colony,WorkerType.LEGENDARY_RUNNER,new Resourses(0,1,1,1),true,new
List<Modificator>()
                {
                 Modificator.CantBeAttackedFirst
                });
        }

        //
        //   WARIOUR CREATIONS
        //
        public static Wariour Create_Usual_Wariour(Colony colony)
        {
            return new Wariour("Обычный", 1, 0, 1,colony,WariourType.USUAL, 1, 1);
        }

        public static Wariour Create_Advanced_Wariour(Colony colony)
        {
            return new Wariour("Продвинутый",6,2,4,colony,WariourType.ADVANCED,2,1);
        }

        public static Wariour Create_AdvancedRevengefull_Wariour(Colony colony)
        {
            return new Wariour("Продвинутый
мстительный",6,2,4,colony,WariourType.ADVANCED_REVENGEFULL,2,1,new
List<Modificator>()
                {
                    Modificator.KillKiller
                });
        }

        public static Wariour Create_Older_Wariour(Colony colony)
        {
            return new Wariour("Старший", 2, 1, 2,colony,WariourType.OLDER ,1, 1);
        }

        public static Wariour Create_Elite_Wariour(Colony colony)
        {
            return new Wariour("Элитный",8,4,3,colony,WariourType.ELITE,2,2);
        }

        public static Wariour Create_LegendaryFat_Wariour(Colony colony)
        {
            return new Wariour("Легендарный
толстый",10,6,4,colony,WariourType.LEGENDARY_FAT,3,1,new List<Modificator>()
                {
                    Modificator.Provocation
                ,Modificator.DoubleHPDEF});
        }
        //
        //   SPECIALINSECT CREATION
        //
        public static SpecialInsect Create_SpecialInsec_Orange(Colony colony)
        {
            return new SpecialInsect("Ленивый обычный агрессивный настойчивый -
Шмель", 25, 6, 8, colony, 2, 1,
                new List<Modificator>()
                {
                    Modificator.BytesAnyway
                });

        }
        public static SpecialInsect Create_SpecialInsec_Black(Colony colony)
```

```
        {
            return new SpecialInsect("Ленивый неуязвимый агрессивный точный -
Толстоножка", 25, 6, 8, colony, 2, 3,
                new List<Modificator>()
                {
                    Modificator.CantBeAttacked,
                    Modificator.IgnoreDef,
                    Modificator.CanDamageImmortals
                });
        }


        //
        //   QUEENS CREATION
        //
        public static Queen Create_Orange_Queen(Colony colony)
        {
            return new Queen("Шарлотта", 17, 8, 29, colony,
                new int[2] {2, 3}, new int[2] {1, 5},new List<WariourType>()
{WariourType.USUAL,WariourType.ADVANCED,WariourType.ADVANCED_REVENGEFULL},
                new List<WorkerType>()
{WorkerType.USUAL,WorkerType.OLDER,WorkerType.OLDER_UNIQUE});
        }

        public static Queen Create_Black_Queen(Colony colony)
        {
            return new Queen("Шарлотта", 16, 9, 23, colony,new int[2] {3, 4}, new
int[2] {2, 5},
                new List<WariourType>()
{WariourType.OLDER,WariourType.ELITE,WariourType.LEGENDARY_FAT},
                new List<WorkerType>()
{WorkerType.USUAL,WorkerType.LEGENDARY,WorkerType.LEGENDARY_RUNNER});
        }

        //
        //   HEAPS CREATION
        //

        public static List<Heap> Create_Heaps()
        {
            return new List<Heap>()
            {
                new Heap(10, 17, 31, 45,1),
                new Heap(12, 18, 0, 29,2),
                new Heap(33, 18, 23, 0,3),
                new Heap(29, 0, 0, 15,4),
                new Heap(13, 34, 0, 39,5)
            };
        }
        //
        //    COLONY FIRST CREATION
        //
        public static Colony Create_Orange_Colony()
        {
            Colony colony = new Colony();
            colony.Queen = Create_Orange_Queen(colony);
            //Wariours
            for (int i = 0; i < 8; i++)
            {
                int tmp = _random.Next(0, 3);
                switch (tmp)
                {
                    case 0: colony.units_Wariour.Add(Create_Usual_Wariour(colony));
                        break;
```

```
                    case
1:colony.units_Wariour.Add((Create_Advanced_Wariour(colony)));
                        break;
                    case 2:
colony.units_Wariour.Add(Create_AdvancedRevengefull_Wariour(colony));
                        break;
                }
            }
            //Workers
            for (int i = 0; i < 14; i++)
            {
                int tmp = _random.Next(0, 3);
                switch (tmp)
                {
                    case 0: colony.units_Worker.Add(Create_Usual_Worker(colony));
                        break;
                    case 1: colony.units_Worker.Add(Create_Older_Worker(colony));
                        break;
                    case 2:
colony.units_Worker.Add(Create_OlderUnique_Worker(colony));
                        break;
                }
            }

            colony.ColonyType = ColonyTypes.Orange;
            colony.SpecialInsect.Add(Create_SpecialInsec_Orange(colony));
            colony.name = "РЫЖИЕ";
            return colony;
        }
        public static Colony Create_Black_Colony()
        {
            Colony colony = new Colony();
            colony.Queen = Create_Black_Queen(colony);
            //Wariours

            for (int i = 0; i < 5; i++)
            {
                int tmp = _random.Next(0, 3);
                switch (tmp)
                {
                    case 0: colony.units_Wariour.Add(Create_Older_Wariour(colony));
                        break;
                    case 1:colony.units_Wariour.Add(Create_Elite_Wariour(colony));
                        break;
                    case 2:
colony.units_Wariour.Add(Create_LegendaryFat_Wariour(colony));
                        break;
                }
            }
            //Workers
            for (int i = 0; i < 15; i++)
            {
                int tmp = _random.Next(0, 3);
                switch (tmp)
                {
                    case 0: colony.units_Worker.Add(Create_Usual_Worker(colony));
                        break;
                    case 1: colony.units_Worker.Add(Create_Legendary_Worker(colony));
                        break;
                    case 2:
colony.units_Worker.Add(Create_LegendaryRunner_Worker(colony));
                        break;
                }
```

```
            }

            colony.ColonyType = ColonyTypes.Black;
            colony.SpecialInsect.Add(Create_SpecialInsec_Black(colony));
            colony.name = "ЧЕРНЫЕ";
            return colony;
        }
        //
        //  FRIENDLY COLONY CREATIONS
        //
        public static int amountColoniesOrange = 1;
        public static int amountColoniesBlack = 1;
        public static Colony FriendlyColony_Creation(Colony _colony)
        {
            Colony colony = new Colony();
            if (_colony.ColonyType==ColonyTypes.Orange)
            {
                colony.name = $"РЫЖИЕ {amountColoniesOrange}";
                amountColoniesOrange++;
                colony.Queen = Create_Orange_Queen(colony);
                colony.Queen.Queens = new int[2] {0, 0};
                colony.ColonyType = _colony.ColonyType;
                colony.FriendlyColonies.Add(_colony);
            }
            else
            {
                colony.name = $"ЧЕРНЫЕ {amountColoniesBlack}";
                amountColoniesBlack++;
                colony.Queen = Create_Black_Queen(colony);
                colony.Queen.Queens = new int[2] {0, 0};
                colony.ColonyType = _colony.ColonyType;
            }
            _colony.FriendlyColonies.Add(colony);
            return colony;
        }

       public static List<string> print_Modificators(ICanGoToHeap unit)
        {
            List<string> mods = new List<string>();
            foreach (Modificator modificator in unit.ModificatorsReturn())
            {
                switch (modificator)
                {
                    case Modificator.Immortal:
                    {
                        mods.Add("полностью неуязвим для всех атак (даже смертельных
для неуязвимых)");
                        break;
                    }
                    case Modificator.ModIgnoting:
                    {
                        mods.Add("игнорирует все модификаторы врагов");
                        break;
                    }
                    case Modificator.FindResAnywhere:
                    {
                        mods.Add("всегда находит нужный ресурс в куче, даже если его
больше нет");
                        break;
                    }
                    case Modificator.CantBeAttackedFirst:
                    {
                        mods.Add("не может быть атакован первым");
                        break;
```

```
                    }
                    case Modificator.KillKiller:
                    {
                        mods.Add("убивает своего убийцу, даже если он неуязвим");
                        break;
                    }
                    case Modificator.Provocation:
                    {
                        mods.Add("принимает все атаки на себя");
                        break;
                    }
                    case Modificator.DoubleHPDEF:
                    {
                        mods.Add("здоровье и защита увеличены в двое");
                        break;
                    }
                    case Modificator.BytesAnyway:
                    {
                        mods.Add("всегда наносит укус, даже если был убит");
                        break;
                    }
                    case Modificator.CantBeAttacked:
                    {
                        mods.Add("не может быть атакован войнами");
                        break;
                    }
                    case Modificator.IgnoreDef:
                    {
                        mods.Add("игнорирует защиту");
                        break;
                    }
                    case Modificator.CanDamageImmortals:
                    {
                        mods.Add("может наносить урон неуязвимым насекомым");
                        break;
                    }
                }
            }

            return mods;
        }

       public static void print_About_Workers(WorkerType type)
        {
           switch (type)
           {
              case WorkerType.USUAL:
              {
                  Console.WriteLine(
                      $"Тип: Обычный \n"+
                      $"---Параметры: здоровье= 1, защита= 0, может брать 1 ресурс:
'веточка' за раз");
                  break;
              }
              case WorkerType.OLDER:
              {
                  Console.WriteLine(
                      $"Тип: Старший \n"+
                      $"---Параметры: здоровье= 2, защита= 1, может брать 1 ресурс:
'веточка или веточка' за раз");
                  break;
              }
              case WorkerType.OLDER_UNIQUE:
              {
```

```
                Console.WriteLine(
                    $"Тип: Старший неповторимый \n"+
                    $"---Параметры: здоровье= 2, защита= 1, может брать 1 ресурс:
'камушек или росинка' за раз \n" +
                    $"---Модификаторы: полностью неуязвим от всех атак(даже
смертельных для неуязвимых, игнорирует все модификаторы врагов, всегда находит нужный
ресурс в куче(даже если его там нет) ");
                break;
            }
            case WorkerType.LEGENDARY:
            {
                Console.WriteLine(
                    $"Тип: Легендарный \n"+
                    $"---Параметры: здоровье= 10, защита= 6, может брать 3 ресурса:
'листик и камушек и росинка' за раз ");
                break;
            }
            case WorkerType.LEGENDARY_RUNNER:
            {
                Console.WriteLine(
                    $"Тип: Легендарный спринтер \n"+
                    $"---Параметры: здоровье= 10, защита= 6, может брать 3 ресурса:
'листик и камушек и росинка' за раз \n" +
                    $"---Модификаторы: не может быть атакован первым ");
                break;
            }
        }
    }

    public static void print_About_Wariours(WariourType type)
    {
        switch (type)
        {
            case WariourType.USUAL:
            {
                Console.WriteLine(
                    $"Тип: Обычный \n"+
                    $"---Параметры: здоровье=1, защита=0, урон=1, может атаковать
1 цель за раз и наносит 1 укус");
                break;
            }
            case WariourType.ADVANCED:
            {
                Console.WriteLine(
                    $"Тип: Продвинутый \n"+
                    $"---Параметры: здоровье=6, защита=2, урон=4, может атаковать
2 цели за раз и наносит 1 укус");
                break;
            }
            case WariourType.ADVANCED_REVENGEFULL:
            {
                Console.WriteLine(
                    $"Тип: Продвинутый мстительный \n"+
                    $"---Параметры: здоровье=6, защита=2, урон=4, может атаковать
2 цели за раз и наносит 1 укус \n" +
                    $"---Модификаторы: убивает своего убийцу, даже если он
неуязвим");
                break;
            }
            case WariourType.OLDER:
            {
                Console.WriteLine(
                    $"Тип: Старший \n"+
```

```
                        $"---Параметры: здоровье=2, защита=1, урон=2, может атаковать
1 цель за раз и наносит 1 укус");
                    break;
                }
                case WariourType.ELITE:
                {
                    Console.WriteLine(
                        $"Тип: Элитный \n"+
                        $"---Параметры: здоровье=8, защита=4, урон=3, может атаковать
2 цели за раз и наносит 2 укуса");
                    break;
                }
                case WariourType.LEGENDARY_FAT:
                {
                    Console.WriteLine(
                        $"Тип: Легендарный толстый \n"+
                        $"---Параметры: здоровье=10, защита=6, урон=4, может атаковать
3 цели за раз и наносит 1 укус \n" +
                        $"---Модификаторы: принимает все атаки на себя, здоровье и
защита увеличены в двое");
                    break;
                }
            }
        }
    }

public static class DayController
    {
        public delegate void Screen_1_print();

        public static event Screen_1_print screen_1;

        public static void Screen_1()
        {
            screen_1.Invoke();
        }
        public delegate void Screen_2_print();

        public static event Screen_2_print screen_2;

        public static void Screen_2()
        {
            screen_2.Invoke();
        }
        public delegate void Screen_3_HeakStart();

        public static event Screen_3_HeakStart screen_3_HeakStart;

        public static void Screen_3_Start()
        {
            screen_3_HeakStart.Invoke();
        }
        //-------------GoToHeap------------
        public delegate void HeapGoing();

        public static event HeapGoing heapGoing;

        public static void GoToHeapStart()
        {
            heapGoing.Invoke();
        }
        //-------------HeapEvent(Fights,TakeRes)---------------
        public delegate void StartHeapEVENT();
```

```
        public static event StartHeapEVENT startHeapEvent;

        public static void HeapEvent()
        {
            startHeapEvent.Invoke();
        }
        //---------------DeleteDeadAnts-----------------------
        public delegate void DeleteDeadUNITS_BornIns_Screen3(int day);

        public static event DeleteDeadUNITS_BornIns_Screen3 DeleteDead_Borning_print;

        public static void DeletingAnts_BornIns_printScreen3(int day)
        {
            DeleteDead_Borning_print.Invoke(day);

        }
    }

public class Squirrel
    {
        public int Duration;
        public int DayOfAttack;
        public bool attacking;

        public Squirrel()
        {
            Duration = 7;
            DayOfAttack = 0;
            attacking = false;
        }

        public void SquirrelAttacky(int Day)
        {
            if (!attacking)
            {
                int tmp = Globals._random.Next(0, 3);
                if (tmp == 0)
                {
                    attacking = true;
                    DayOfAttack = Day;
                }
            }
            if (DayOfAttack + Duration == Day && attacking)
            {
                attacking = false;
            }
            if (attacking)
            {
                bool open = true;
                while (open)
                {
                    int tmp = Globals._random.Next(0, Heaps.heaps.Count);
                    if (Heaps.heaps[tmp].IsOpen)
                    {
                        Heaps.heaps[tmp].amountOfRes[0] -=
(Heaps.heaps[tmp].amountOfRes[0]) / 2;
                        open = false;
                    }
                }
            }
        }
    }

}
```