

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Курсовая работа

По дисциплине: «Объектно-ориентированное программирование»

На тему “Симулятор мира”

Выполнил: студент 2 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверила: ассистент кафедры ПМиК

Суходоева Н. Н.

Новосибирск, 2018

**Содержание**

1. Постановка задачи ...........................................................................................4

2. Иерархия классов..............................................................................................5

3. Описание алгоритма основной программы ...................................................7

4. Личный вклад ...................................................................................................8

5. Результаты работы ...........................................................................................9

6. Заключение ......................................................................................................11

**Постановка задачи**

Программа должна быть написана с использованием объектно-ориентированных технологий. Описания объектов и методов необходимо оформить в отдельном модуле. Допускаются и поощряются различные творческие модификации по существу расширяющие и развивающие реализацию задания.

Необходимый минимум содержания работы:

Инкапсуляция (все поля данных не доступны из внешних функций)

Наследование (минимум 3 класса, один из которых - абстрактный)

Полиморфизм

Конструкторы, Перегрузка конструкторов

Списки инициализации

Желательно: минимум + 2 технологии ООП (статические элементы, дружественные функции, классы, виртуальные функции, шаблоны, множественное наследование, массивы указателей на объекты, конструкторы копирования, параметры по умолчанию, использование объектов в качестве аргументов или возвращаемых значений)

С помощью ООП разработать возможную реализацию некоторого компьютерного мира. Обитатели такого мира могут иметь различную форму, быть подвижными и неподвижными, быстрыми и медленными, могут размножаться, скрещиваться, нападать и защищаться и т.д. (выполнять могут до трех человек в группе) Работа была выполнена совместно со студентом группы ИП-711 Щерба А.С.

**Структура классов**

**SupportClass**

**protected:**

static int ScreenHeigth

static int ScreenWidth

static listc NonGo

static listc SlowGo

static listc FastGo

static int size\_of\_cube

float const\_timer\_step

struct radius

{

Int x;

Int y;

}\*rad

**public:**

struct XY

{

int mode

float x

float y

}

static void push\_NonGo(int i)

static void push\_SlowGo(int i)

static void SetScreen(int s\_h = 720, int s\_w = 1280)

static void SetCube(int s =20)

static void GetCube()

int retSH()

int ret SW()

**Listc**

**Protected:**

Struct list

{

Int id

Int weight

Int value

List\* next

List\* prev

}

List \*head, \*tail

Int length

**Public:**

Listc(int I, int v) : head(NULL), tail(NULL)

Listc() : head(NULL), tail(NULL)

~listc()

Void setLength(int x)

Void setHead(list\* x)

Void setTail(list\* x)

Int getLength()

List\* getHead()

List\* getTail()

Virtual void push(int I, int v = 0, int w = 0)

Virtual void output()

List\* search(int i)

Void DeleteList()

Inventory

**Private:**

Int Weigth

**Public:**

Inventory()

Int weight()

Void push(int I, int v, int w) override

Void output() override

List\* take(int l)

**Object**

**Protected:**

int hp

Int id

Int sq\_x

Int sq\_y

Double x

Double y

Texture texture

Sprite sprite

Inventory Inv

object(const object &copy)

**Public:**

Object(){}

Virtual ~object()

Virtual object\* copy() const = 0

Virtual void f() = 0

Sprite getSprite()

Int ID()

Virtual void reload()

Inventory \*inv()

Float retX()

Float retY()

XY masCoord()

Int getHP()

Void setHP(int I = 10)

**Fauna**

**Protected:**

String sex\_t[2];

float base\_move;

int hunger;

int radar;

int mating\_time;

int to\_x;

int to\_y;

float step\_x;

float step\_y;

int sex;

bool isAttack;

bool isMating;

bool isMove;

bool isNon;

bool isFear;

bool isSelect;

float timer\_mating;

float timer\_step;

listc ration;

fauna(const fauna &copy) : object(copy)

**Public:**

Virtual void f() = 0

Virtual ~fauna()

fauna() : timer\_step(0), hunger(0), radar(8)

Void changeStep(float step)

Virtual void goTO(int t\_x, int t\_y, int s\_x, int s\_y)

Virtual void Move(float time1)

Bool ismove()

Bool ismove(bool x)

Bool isselect()

Bool isselect(bool x)

**flora**

**private:**

static int i

String growing[5]

Int time\_to\_break

Int lvl\_gr

Int maxgr

Bool isGrow

**public:**

static int getI()

static int setI(int ch, int ii)

flora(){}

flora(String s, int ID, int xe, int ye, int xe\_s, int ye\_s)

flora(const flora &copy) : object(copy)

virtual flora\* copy() const

void grow()

void take()

bool isgrow()

bool isgrow(bool x)

void f()

**nonOrg**

**private:**

static int i

int time\_to\_break

String back

**public:**

static int getI()

nonOrg(){}

nonOrg(String s, int ID, int xe, int ye, int xe\_s, int ye\_s)

nonOrg(const nonOrg &copy) : object(copy)

nonOrg\* copy() const

void reload()

void f()

**village**

**private:**

static int i

String back

**public:**

static int getI()

village(){}

village(String s, int ID, int xe, int ye, int xe\_s, int ye\_s)

village(const village &copy) : object(copy)

village\* copy() const

void f()

**Object**

**animal**

**Private:**

Static int i

**Public:**

Static int getI()

Static int setI(int ch, int ii)

animal()

animal(String \*s, int ID, int hitp, int xe, int ye, int xe\_s, int ye\_s, int p)

animal(const animal &copy) : fauna(copy)

animal \* copy() const

Void f()

**beast**

**Private:**

Static int i

**Protected:**

Int dam

**Public:**

Static int getI()

Static int setI(int ch, int ii)

beast()

beast(String \*s, int ID, int hitp, int xe, int ye, int xe\_s, int ye\_s, int p)

beast(const beast &copy) : fauna(copy)

beast\* copy() const

int Dam()

void Dam(int i)

Void f()

**Human**

**Private:**

Static int i

**Public:**

Static int getI()

Static int setI(int ch, int ii)

Human()

Human(String \*s, int ID, int hitp, int xe, int ye, int xe\_s, int ye\_s, int p)

human(const human &copy) : fauna(copy)

human\* copy() const

Void f()

**Описание алгоритма основной программы**

Основная программа подключает необходимые для её функционирования библиотеки (SFML и стандартные iostream). Затем создаётся объект окна с разрешением 1280x720 пикселей. Подключает модуль с описанием класса.

В файле world\_objects.h полностью реализована иерархия классов, в файле map.h находятся карта мира и карта объектов.

В файле OOP\_Project.cpp рисуется карта, создаются объекта классов human, beast, animal, flora. Затем происходит отрисовка этих объектов на карте (рис. 1), после чего они начинают взаимодействовать с картой и друг с другом. Есть возможность менять их движение с помощью мышки.

Мир имеет два логических варианта окончания:

1.На карте остались какие – либо мирные животные (мышь, олень, кролик) (рис. 2)

2.Все живые существа погибли (рис. 3)

**Личный вклад**

Были выполнены:

построение иерархии классов;

реализация классов flora, fauna и SupportClass;

реализация функций движения для объектов;

создание массивов указателей на объекты и их инициализация;

поведение объектов на карте;

**Результаты работы**

****

Рис. 1 - Начальный мир



Рис.2 – на карте осталось два оленя

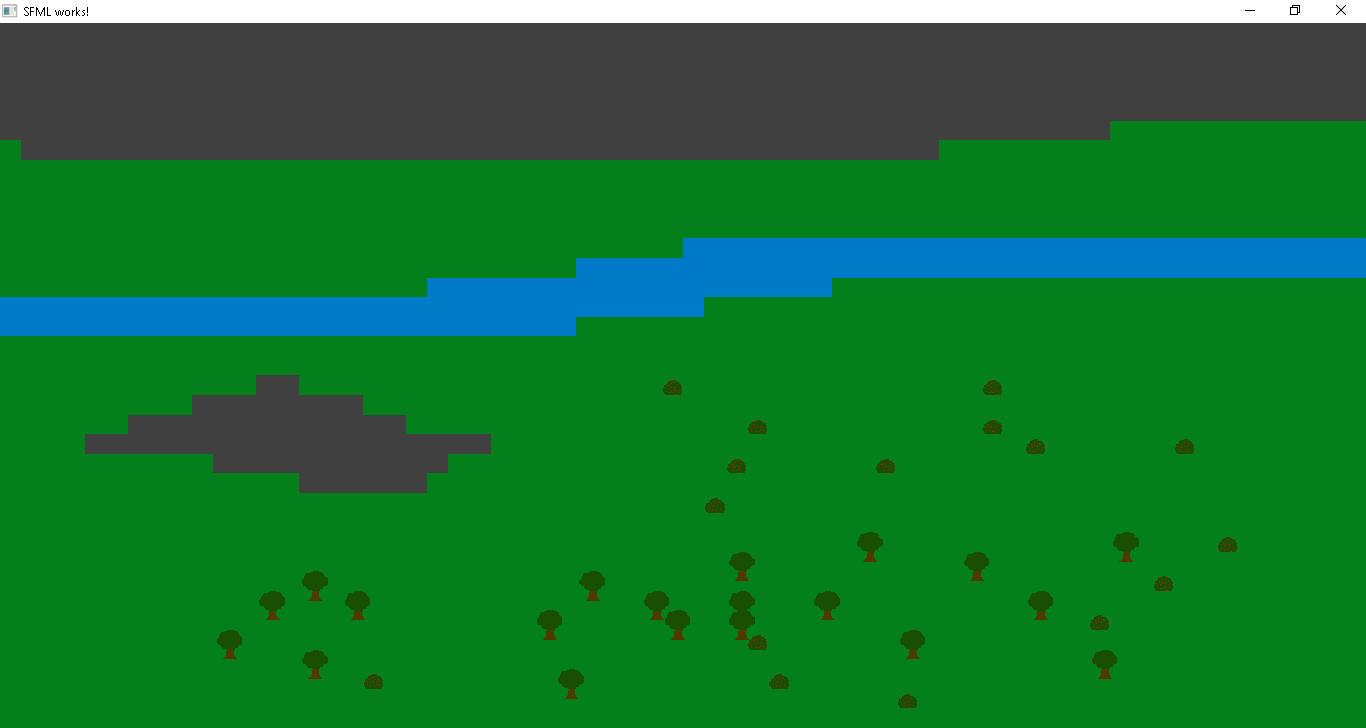
****

Рис. 3 - Все живые существа погибли

**Заключение**

При работе использовались такие принципы ООП, как наследование, полиморфизм и инкапсуляция.

Инкапсуляция – механизм скрытия данных внутри объекта.

Наследование – конструирование новых, более сложных производных классов-потомков, из уже имеющихся базовых классов-родителей, с помощью добавления новых полей и/или методов.

Полиморфизм - механизм, обеспечивающий возможность определения различных описаний некоторого единого метода (единого по названию) для классов различного уровня иерархии. Для создания объектов классов в программе существуют конструкторы, была реализована их перегрузка. Также использовались чистые виртуальные методы и абстрактный класс.

Абстрактный класс – класс, в котором есть хотя бы один чистый (обнуленный) виртуальный метод. Объекты таких классов создавать запрещено. Он служит основой для производных классов.

Также использовались статические элементы, виртуальные функции, массивы указателей на объекты и конструкторы копирования.