Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Петрозаводский государственный университет» Физико-технический институт Кафедра информационно-измерительных систем и физической электроники

**ОТЧЕТ**

по командному заданию

по курсу «Технология программирования»

тема «Командная разработка приложения на языке программирования С++

с использованием SFML-библиотек»

Авторы работы:

студенты 21316 группы

Едигарев Григорий,

Кулакова Мария,

Кулагин Александр

Преподаватель:

Бульба Артем Владимирович

Петрозаводск 2018

**Цель работы:** На C++ (в Visual Studio 2010 Express) с использованием SFML (Simple and Fast Multimedia Library — простая и быстрая мультимедийная библиотека) написать простую 2D игру.

**Краткое словесное описание сюжета игры:**

Искатель приключений попадает в подземелье, которое представляет из себя простой лабиринт с агрессивно настроенными скелетами. Герою предстоит собрать сокровища и остаться в живых в этих опасных условиях, и для выживания ему поможет его смелость, упорство, а также умение пользоваться огнестрельным оружием.

**Кратко о программной реализации:**

Среда разработки - Microsoft Visual Studio 2010.

Язык - C++ с использованием SFML.

При создании игры были созданы два файла:

map.h - заголовочный файл с кодом карты

main.cpp - главный файл исходного кода

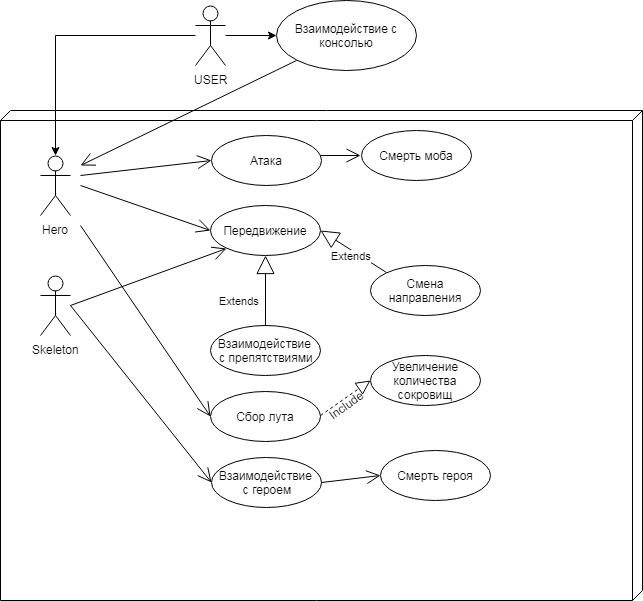
Детальное описание функционала главного файла:

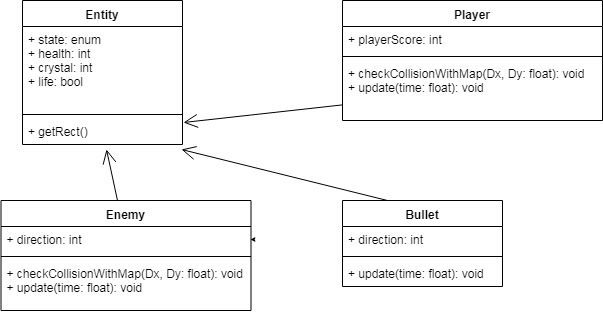
* подключение необходимых библиотек
* класс сущность
* класс игрока
* класс врага
* класс пули
* меню, рестарт игры, запуск

**Заключение.**

Написали программу на языке C++ с использованием библиотеки SFML и системы контроля версий Git. Все прецеденты реализовали. Игра имеет некоторые сбои. Принципы раздельной компиляции использованы (main.cpp и map.h). Очистка памяти реализована за счет удаления элементов из списка. Приложены диаграмма прецедентов и диаграмма классов. Поставленная цель достигнута.

**Приложение (диаграммы, исходный код программы).**





main.cpp

#include <iostream>

#include <sstream>

#include <SFML/Graphics.hpp>

#include <SFML/Audio.hpp>

#include "map.h" //подключили код с картой

#include <list>

using namespace sf;

////////////////////////////КЛАСС СУЩНОСТЬ////////////////////////

class Entity {

public:

enum { left, right, up, down, stay} state;// тип перечисления - состояние объекта

float dx, dy, x, y, speed, moveTimer;//добавили переменную таймер для будущих целей

int w, h, health, crystal; //переменная “health”, хранящая жизни игрока

bool life; //переменная “life” жизнь, логическая

Texture texture;//сфмл текстура

Sprite sprite;//сфмл спрайт

float CurrentFrame;//хранит текущий кадр

std::string name;//враги могут быть разные, врагов можно различать по именам //каждому можно дать свое действие в update() в зависимости от имени

Entity(Image &image, float X, float Y, int W, int H, std::string Name){

x = X; y = Y; //координата появления спрайта

w = W; h = H;

name = Name;

moveTimer = 0;

dx = 0; dy = 0;

speed = 0;

CurrentFrame = 0;

health = 100;

life = true; //инициализировали логическую переменную жизни, герой жив

texture.loadFromImage(image); //заносим наше изображение в текстуру

sprite.setTexture(texture); //заливаем спрайт текстурой

}

FloatRect getRect(){

return FloatRect(x, y, w, h);

}

/\*Тип данных (класс) "sf::FloatRect" позволяет хранить четыре координаты прямоугольника//в нашей игре это координаты текущего расположения тайла на карте

далее это позволит спросить, есть ли ещё какой-либо тайл на этом месте //эта ф-ция нужна для проверки пересечений \*/

virtual void update(float time) = 0;

};

////////////////////////////КЛАСС ИГРОКА////////////////////////

class Player :public Entity {

public:

int takeHp, playerScore, KillEn;//эта переменная может быть только у игрока

Player(Image &image, float X, float Y, int W, int H, std::string Name) :Entity(image, X, Y, W, H, Name){

playerScore = 0;

KillEn = 0;

takeHp = 0;

state = stay;

if (name == "Player"){ //Задаем спрайту один прямоугольник для //вывода одного игрока. IntRect – для приведения типов

sprite.setTextureRect(IntRect(0, 0, w, h));

}

}

void control(){

if (Keyboard::isKeyPressed(Keyboard::Left)) {

state = left;

speed = 0.2;

}

if (Keyboard::isKeyPressed(Keyboard::Right)) {

state = right;

speed = 0.2;

}

if (Keyboard::isKeyPressed(Keyboard::Up)) {

state = up;

speed = 0.2;

}

if (Keyboard::isKeyPressed(Keyboard::Down)) {

state = down;

speed = 0.2;

}

}

//Метод проверки столкновений с элементами карты

void checkCollisionWithMap(float Dx, float Dy) {

for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты

for (int j = x / 32; j<(x + w) / 32; j++)

{

if (TileMap[i][j] == '0')//если элемент тайлик земли

{

if (Dy > 0) { y = i \* 32 - h; dy = 0; }//по Y

if (Dy < 0) { y = i \* 32 + 32; dy = 0; }//столкновение с верхними краями

if (Dx > 0) { x = j \* 32 - w; dx = 0; }//с правым краем карты

if (Dx < 0) { x = j \* 32 + 32; dx = 0; }// с левым краем карты

}

if (TileMap[i][j] == 'h') {

takeHp++;//если взяли сердечко

TileMap[i][j] = ' ';//убрали сердечко

}

if (TileMap[i][j] == 'c') {

playerScore++;//если взяли сердечко

TileMap[i][j] = ' ';//убрали сердечко

}

}

}

void update(float time) //метод "оживления/обновления" объекта класса.

{

if (life) {//проверяем, жив ли герой

control();//функция управления персонажем

switch (state)//делаются различные действия в зависимости от состояния

{

case right:{//состояние идти вправо

dx = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 192, 96, 96));

break;

}

case left:{//состояние идти влево

dx = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 96, 96, 96));

break;

}

case up:{//идти вверх

dy = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 288, 96, 96));

break;

}

case down:{//идти вниз

dy = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(96 \* int(CurrentFrame), 0, 96, 96));

break;

}

case stay:{//стоим

dy = speed;

dx = speed;

break;

}

}

x += dx\*time; //движение по “X”

checkCollisionWithMap(dx, 0);//обрабатываем столкновение по Х

y += dy\*time; //движение по “Y”

checkCollisionWithMap(0, dy);//обрабатываем столкновение по Y

speed = 0; //обнуляем скорость, чтобы персонаж остановился. //state = stay;

sprite.setPosition(x, y); //спрайт в позиции (x, y).

if (health <= 0){ life = false; }//если жизней меньше 0, либо равно 0, то умираем

}

}

};

////////////////////////////КЛАСС ВРАГА////////////////////////

class Enemy :public Entity{

public:

int direction;//направление движения врага

Enemy(Image &image, float X, float Y, int W, int H, std::string Name) :Entity(image, X, Y, W, H, Name){

if (name == "EasyEnemy"){

//Задаем спрайту один прямоугольник для

//вывода одного игрока. IntRect – для приведения типов

sprite.setTextureRect(IntRect(0, 0, w, h));

direction = rand() % (3); //Направление движения врага задаём случайным образом

//через генератор случайных чисел

speed = 0.25;//даем скорость.этот объект всегда двигается

dx = speed;

}

}

void checkCollisionWithMap(float Dx, float Dy)//ф-ция проверки столкновений с картой

{

for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты

for (int j = x / 32; j<(x + w) / 32; j++)

{

if (TileMap[i][j] == '0')//если элемент - тайлик земли

{

if (Dy > 0) {

y = i \* 32 - h; dy = -0.1;

direction = rand() % (3); //Направление движения врага

}//по Y

if (Dy < 0) {

y = i \* 32 + 32; dy = 0.1;

direction = rand() % (3);//Направление движения врага

}//столкновение с верхними краями

if (Dx > 0) {

x = j \* 32 - w; dx = -0.1;

direction = rand() % (3);//Направление движения врага

}//с правым краем карты

if (Dx < 0) {

x = j \* 32 + 32; dx = 0.1;

direction = rand() % (3); //Направление движения врага

}// с левым краем карты

}

}

}

void update(float time)

{

if (name == "EasyEnemy"){//для персонажа с таким именем логика будет такой

if (life) {//проверяем, жив ли герой

switch (direction)//делаются различные действия в зависимости от состояния

{

case 0:{//состояние идти вправо

dx = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(60 \* int(CurrentFrame), 120, 50, 50));

break;

}

case 1:{//состояние идти влево

dx = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(50 \* int(CurrentFrame), 60, 50, 60));

break;

}

case 2:{//идти вверх

dy = -speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(50 \* int(CurrentFrame), 180, 50, 60));

break;

}

case 3:{//идти вниз

dy = speed;

CurrentFrame += 0.005\*time;

if (CurrentFrame > 3) CurrentFrame -= 3;

sprite.setTextureRect(IntRect(50 \* int(CurrentFrame), 0, 50, 60));

break;

}

}

x += dx\*time; //движение по “X”

checkCollisionWithMap(dx, 0);//обрабатываем столкновение по Х

y += dy\*time; //движение по “Y”

checkCollisionWithMap(0, dy);//обрабатываем столкновение по Y

sprite.setPosition(x, y); //спрайт в позиции (x, y).

if (health <= 0){ life = false; }//если жизней меньше 0, либо равно 0, то умираем

}

}

}

};//класс Enemy закрыт

////////////////////////////КЛАСС ПУЛИ////////////////////////

class Bullet :public Entity{//класс пули

public:

int direction;//направление пули.всё так же, только взяли в конце состояние игрока (int dir) для задания направления полёта пули

Bullet(Image &image, float X, float Y, int W, int H, std::string Name, int dir) :Entity(image, X, Y, W, H, Name){

x = X;

y = Y;

direction = dir;

speed = 0.5;

w = h = 16;

life = true;

//выше инициализация в конструкторе

}

void update(float time)

{

switch (direction)

{

case 0: dx = -speed; dy = 0; break;// state = left

case 1: dx = speed; dy = 0; break;// state = right

case 2: dx = 0; dy = -speed; break;// state = up

case 3: dx = 0; dy = speed; break;// state = down

}

if (life){

x += dx\*time;//само движение пули по х

y += dy\*time;//по у

if (x <= 0) x = 20;// задержка пули в левой стене, чтобы при проседании кадров она случайно не вылетела за предел карты и не было ошибки (сервер может тормозить!)

if (y <= 0) y = 20;

if (x >= 800) x = 780;// задержка пули в правой стене, чтобы при проседании кадров она случайно не вылетела за предел карты и не было ошибки (сервер может тормозить!)

if (y >= 640) y = 620;

for (int i = y / 32; i < (y + h) / 32; i++)//проходимся по элементам карты

for (int j = x / 32; j < (x + w) / 32; j++)

{

if (TileMap[i][j] == '0')//если элемент наш тайлик земли, то

life = false;// то пуля умирает

}

sprite.setPosition(x + w / 0.5, y + h / 0.35);//задается позицию пули

}

}

};

////////menu

void menu(RenderWindow & window) {

Texture menuTexture1, menuTexture2, menuTexture3, aboutTexture, menuBackground;

menuTexture1.loadFromFile("images/new\_game.png");

menuTexture2.loadFromFile("images/about\_game.png");

menuTexture3.loadFromFile("images/exit.png");

aboutTexture.loadFromFile("images/verdont.png");

menuBackground.loadFromFile("images/menu.png");

Sprite menu1(menuTexture1), menu2(menuTexture2), menu3(menuTexture3), about(aboutTexture), menuBg(menuBackground);

bool isMenu = 1;

int menuNum = 0;

menu1.setPosition(100, 30);

menu2.setPosition(100, 90);

menu3.setPosition(100, 150);

menuBg.setPosition(345, 0);

//////////////////////////////МЕНЮ///////////////////

while (isMenu)

{

menu1.setColor(Color::White);

menu2.setColor(Color::White);

menu3.setColor(Color::White);

menuNum = 0;

window.clear(Color(0, 0, 0));

if (IntRect(10, 30, 300, 50).contains(Mouse::getPosition(window))) { menu1.setColor(Color::Red); menuNum = 1; }

if (IntRect(10, 90, 300, 50).contains(Mouse::getPosition(window))) { menu2.setColor(Color::Yellow); menuNum = 2; }

if (IntRect(10, 150, 300, 50).contains(Mouse::getPosition(window))) { menu3.setColor(Color::Green); menuNum = 3; }

if (Mouse::isButtonPressed(Mouse::Left))

{

if (menuNum == 1) isMenu = false;//если нажали первую кнопку, то выходим из меню

if (menuNum == 2) { window.draw(about); window.display(); while (!Keyboard::isKeyPressed(Keyboard::Escape)); }

if (menuNum == 3) { window.close(); isMenu = false; }

}

window.draw(menuBg);

window.draw(menu1);

window.draw(menu2);

window.draw(menu3);

window.display();

}

////////////////////////////////////////////////////

}

//рестартыч игры

bool isGameStart() {

sf::VideoMode desktop = sf::VideoMode::getDesktopMode();

sf::RenderWindow window(sf::VideoMode(800, 600, desktop.bitsPerPixel), "Skeleton");

menu(window); //menu

Font font;//шрифт

font.loadFromFile("ttf/CyrilicOld.ttf");//передаем нашему шрифту файл шрифта

Text text("", font, 20);//создаем объект текст

text.setColor(Color::Green);//покрасили текст в color text.setStyle(Text::Bold);//жирный текст.

//на звукичах

SoundBuffer shootBuffer;//создаём буфер для звука

shootBuffer.loadFromFile("audio/shoot.ogg");//загружаем в него звук

Sound shoot(shootBuffer);//создаем звук и загружаем в него звук из буфера

shoot.setVolume(20);

SoundBuffer g\_oBuffer;//создаём буфер для звука

g\_oBuffer.loadFromFile("audio/g\_o.ogg");//загружаем в него звук

Sound g\_o(g\_oBuffer);//создаем звук и загружаем в него звук из буфера

g\_o.setVolume(20);

Music music;//создаем объект музыки

music.openFromFile("audio/music.ogg");//загружаем файл

music.setVolume(30.f);

music.play();//воспроизводим музыку

music.setLoop(true);

Image map\_image;//объект изображения для карты

map\_image.loadFromFile("images/map.png");//загружаем файл для карты

Texture map;//текстура карты

map.loadFromImage(map\_image);//заряжаем текстуру картинкой

Sprite s\_map;//создаём спрайт для карты

s\_map.setTexture(map);//заливаем текстуру спрайтом

Clock clock;

Clock gameTimeClock;//переменная игрового времени, будем здесь хранить время игры

int gameTime = 0;//объявили игровое время, инициализировали.

Image heroImage;

heroImage.loadFromFile("images/hero.png"); // загружаем изображение игрока

Image easyEnemyImage;

easyEnemyImage.loadFromFile("images/skel.png"); // загружаем изображение врага

Image BulletImage;//изображение для пули

BulletImage.loadFromFile("images/bullet.png");//загрузили картинку в объект изображения

Player p(heroImage, 100, 170, 96, 96, "Player");//объект класса игрока

std::list<Entity\*> entities; //сущности

std::list<Entity\*>::iterator it; //итератор чтобы проходить по элементам списка

std::list<Entity\*>::iterator it2;

const int ENEMY\_COUNT = 1; //максимальное количество врагов в игре

int enemiesCount = 0; //текущее количество врагов в игре

//Заполняем список объектами врагами

for (int i = 0; i < ENEMY\_COUNT; i++)

{

float xr = 150 + rand() % 500; // случайная координата врага на поле игры по оси “x”

float yr = 150 + rand() % 350; // случайная координата врага на поле игры по оси “y”

//создаем врагов и помещаем в список

entities.push\_back(new Enemy(easyEnemyImage, xr, yr, 96, 96, "EasyEnemy"));

enemiesCount += 1; //увеличили счётчик врагов

}

int createObjectForMapTimer = 0;//Переменная под время для генерирования камней

while (window.isOpen())

{

float time = clock.getElapsedTime().asMicroseconds();

if (p.life) gameTime = gameTimeClock.getElapsedTime().asSeconds();//игровое время в

//секундах идёт вперед, пока жив игрок. Перезагружать как time его не надо. оно не обновляет логику игры

clock.restart();

time = time / 1000;

createObjectForMapTimer += time;//наращиваем таймер

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

window.close();

//стреляем по нажатию клавиши "P"

if (event.type == sf::Event::KeyPressed)

{

if (event.key.code == sf::Keyboard::P)

{

entities.push\_back(new Bullet(BulletImage, p.x, p.y, 16, 16, "Bullet", p.state));

shoot.play();//играем звук пули

}

}

}

//game restart

if (Keyboard::isKeyPressed(Keyboard::Tab)) { return true; }//если таб, то перезагружаем игру

if (Keyboard::isKeyPressed(Keyboard::Escape)) { return false; }//если эскейп, то выходим из игры

p.update(time); //оживляем объект “p” класса “Player”

//Проверяем список на наличие "мертвых" пуль и удаляем их

for (it = entities.begin(); it != entities.end();)//говорим что проходимся от начала до конца

{// если этот объект мертв, то удаляем его

if ((\*it)-> life == false) { it = entities.erase(it); }

else it++;//и идем курсором (итератором) к след объекту.

}

//Проверка пересечения игрока с врагами.Если пересечение произошло, то "health = 0", игрок обездвижевается и выводится сообщение "you are lose"

if (p.life == true){//если игрок жив

for (it = entities.begin(); it != entities.end(); it++){//бежим по списку врагов

if ((p.getRect().intersects((\*it)->getRect())) && ((\*it)->name == "EasyEnemy"))

{

p.health = 0;

std::cout << " You are lose!";

p.life == false;

music.setVolume(0);

g\_o.setVolume(80);

g\_o.play();

}

}

}

for (it = entities.begin(); it != entities.end();)//говорим что проходимся от начала до конца

{

Entity \*b = \*it;//для удобства, чтобы не писать (\*it)->

b->update(time);//вызываем ф-цию update для всех объектов (по сути для тех, кто жив)

if (b->life == false) { it = entities.erase(it); delete b; }// если этот объект мертв, то удаляем его

else it++;//и идем курсором (итератором) к след объекту. так делаем со всеми объектами списка

}

//взаимодействие

for (it = entities.begin(); it != entities.end(); it++)//проходимся по эл-там списка

{

if ((\*it)->name == "EasyEnemy")

{

Entity \*enemy = \*it;

for (std::list<Entity\*>::iterator it2 = entities.begin(); it2 != entities.end(); it2++)

{

Entity \*bullet = \*it2;

if (bullet->name == "Bullet")

if (bullet->life == true)

{

if (bullet->getRect().intersects(enemy->getRect()))

{

bullet->life = false;

enemy->life = false;

float xr = 150 + rand() % 500; // случайная координата врага на поле игры по оси “x”

float yr = 150 + rand() % 350; // случайная координата врага на поле игры по оси “y” //создаем врагов и помещаем в список

entities.push\_back(new Enemy(easyEnemyImage, xr, yr, 96, 96, "EasyEnemy"));

p.KillEn++;

}

}

}

}

}

window.clear();

/////////////////////////////Рисуем карту/////////////////////

for (int i = 0; i < HEIGHT\_MAP; i++)

for (int j = 0; j < WIDTH\_MAP; j++)

{

if (TileMap[i][j] == ' ') s\_map.setTextureRect(IntRect(0, 0, 32, 32));

if (TileMap[i][j] == 'c') s\_map.setTextureRect(IntRect(96, 0, 32, 32));

if ((TileMap[i][j] == '0')) s\_map.setTextureRect(IntRect(64, 0, 32, 32));

if ((TileMap[i][j] == 'h')) s\_map.setTextureRect(IntRect(128, 0, 32, 32));//сердце

s\_map.setPosition(j \* 32, i \* 32);

window.draw(s\_map);

}

//объявили переменную здоровья и времени

std::ostringstream gameHp, gameTimeString, gameCrystal, gameKillEn;

gameTimeString << gameTime;//формируем строку

gameCrystal << p.playerScore;

gameKillEn << p.KillEn;

gameHp << p.takeHp;

text.setString("\nВремя игры: " + gameTimeString.str() + "\nСобрано сердец " + gameHp.str() + "\nКристаллы " + gameCrystal.str() + "\nУбито врагов " + gameKillEn.str());//задаем строку тексту

text.setPosition(50, 50);//задаем позицию текста

window.draw(text);//рисуем этот текст

//рисуем врагов

for (it = entities.begin(); it != entities.end(); it++)

{

if ((\*it)->life) //если враги живы

window.draw((\*it)->sprite); //рисуем

}

//рисуем пули

for (it = entities.begin(); it != entities.end(); it++)

{

if ((\*it)->life) //если пули живы

window.draw((\*it)->sprite); //рисуем объекты

}

window.draw(p.sprite);//рисуем спрайт объекта “p” класса “Player”

window.display();

}

};

void gameRunning(){//ф-ция перезагружает игру , если это необходимо

if (isGameStart()) { gameRunning(); }////если startGame() == true, то вызываем занова ф-цию isGameRunning, которая в свою очередь опять вызывает startGame()

}

int main()

{

gameRunning();//запускаем процесс игры

return 0;

}

map.h

#include <SFML\Graphics.hpp>

const int HEIGHT\_MAP = 20;//размер карты высота

const int WIDTH\_MAP = 25;//размер карты ширина

std::string TileMap[HEIGHT\_MAP] = {

"0000000000000000000000000",

"0h c 0 h 0",

"0 0",

"0 0",

"0 0 0",

"0 0 0",

"0 0",

"0 0",

"0 0",

"0 0",

"0 000 0",

"0 0",

"0 0",

"0 h 0",

"0 0",

"0 00000 0",

"0 h 0",

"0 0",

"0c c0",

"0000000000000000000000000",

};