Міністерство освіти і науки України Дніпропетровський технікум зварювання та електроніки ім. Є. О. Патона

Системне програмування

Курсовий проект

*Студента*_____
Групи _____
Перевірила Саприкіна І.Г.

2011 - 2012 н.р.

ЗАВДАННЯ

на курсове проектування з предмету :

«Системне програмування>

студенту спеціальності:

5.091504 «Обслуговування комп'ютерних і інтелектуальних систем та мереж»

курсу група_____

Дніпропетровського технікуму зварювання та електроніки ім. Є.О.Патона

(прізвище та ім я побатькові)

Тема завдання:

ТЕХНІЧНЕ ЗАВДАННЯ НА ПРОЕКТУВАННЯ:

1.Вступ
2.Постановка задачі
3.Опис функціональних можливостей
4.Розробка алгоритму програми
5.Кодування
6.Відлагодження програми
7.Тестування
8.Оформлення проекту

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КУРСОВОГО ПРОЕКТУ

№	Зміст роботи	Орієнтов		Місяці і дні виконання					
1	Постанова задачі	5%							
2	Опис функціональних можливостей	10%							
3	Розробка алгоритму	30%							
4	Лістинг програми	20%							
5	Тестування програми	20%							
6	Результати реалізації	15%							

дата видачі«___»_____2011р.

Строк виконання «___»_____2012р.

Викладач, керівник
курсового проекту_____

Затвердженно предметною комісією протокол №____від«___»_____2011р.
предметної комісії_____/А.О. Грязнова/

Зміст

ВСТУП

Кожна людина час від часу виявляється за ситуації, коли досягнення деякого результату може бути здійснене не єдиним способом. У таких випадках доводиться відшукувати якнайкращий спосіб. Проте в різних ситуаціях якнайкращими можуть бути абсолютно різні рішення. Все залежить від вибраного або заданого критерію. На практиці виявляється, що в більшості випадків поняття «якнайкращий» може бути виражене кількісними критеріями - мінімум витрат, мінімум часу, максимум прибули і так далі Тому можлива постановка математичних завдань відшукання оптимального (optimum - якнайкращий) результату, оскільки принципових відмінностей у відшуканні найменшого або найбільшого значення немає. Завдання на відшукання оптимального рішення називаються завданнями оптимізації. Оптимальний результат, як правило, знаходиться не відразу, а в результаті процесу, званого процесом оптимізації. Вживані в процесі оптимізації методи отримали назву методів оптимізації. Щоб вирішити практичну задачу треба перекласти її математичною мовою, тобто скласти її математичну модель.

Математична модель є стрункою і глибокою сукупністю знань про математичні моделі зі своїми проблемами, з власними шляхами розвитку, обумовленими внутрішніми і зовнішніми причинами і завданнями. Математика дає зручні і плідні способи опису найрізноманітніших явищ реального миру і тим самим виконує в цьому сенсі функцію мови. Цю роль математики чудово усвідомлював Галілей, що сказав: «Філософія написана в грандіозній книзі - Всесвіті, який відкритий нашому пильному погляду. Але зрозуміти цю книгу може лише той, хто навчився розуміти її мову і знаки, якими вона викладена. Написана ж вона на мові математики».

Часто в математичній моделі потрібно знайти найбільше або найменше значення деякої функції на деякій множині, тобто вирішити задачу оптимізації. Методів вирішення завдань оптимізації достатні багато. Деякі з них розглядалися при відшуканні екстремальних значень функцій одній і багатьох речових змінних. Окрім точних методів широко використовуються і наближені, наприклад, метод дихотомії і так далі

Знання методів знаходження оптимального рішення дозволяє інженерові і офіцерові вибирати найбільш ефективні і найекономічніші способи експлуатації і ремонту машин, знаходити оптимальні вирішення тактичних завдань,

Впродовж всієї своєї еволюції чоловік, здійснюючи ці або ІНШІ ДІЯННЯ, прагнув поводитися так, щоб результат, що досягається як наслідок деякого вчинку, опинився в певному значенні якнайкращим. Рухаючись з одного пункту в інший, він прагнув знайти найкоротший серед можливих шлях. Будуючи жито, він шукав таку його геометрію, яка при найменшій витраті палива, забезпечувала прийнятно комфортні умови існування. Займаючись будівництвом кораблів, він намагався надати їм таку форму, при якій вода чинила б найменший опір. Можна легко продовжити перелік подібних прикладів.

Якнайкращі в певному значенні вирішення завдань прийнято нази-

вати оптимальними. Без використання принципів оптимізації в даний час не вирішується жодна більш менш складна проблема. При постановці і вирішенні завдань оптимізації виникають два питання: що і як оптимізувати?

Відповідь на перше питання виходить як результат глибокого вивчення проблеми, яку належить вирішити. Виявляється той параметр, який визначає ступінь досконалості вирішення виниклої проблеми. Цей параметр зазвичай називають цільовою функцією або критерієм якості. Далі встановлюється сукупність величин, які визначають цільову функцію. Нарешті, формулюються всі обмеження, які повинні враховуватися при рішенні задачі. Після цього будується математична модель, що полягає у встановленні аналітичної залежності цільової функції від всіх аргу ментів і аналітичного формулювання супутніх завданню обмежень.

1. Постанова задачі

Скласти тривимірну стратегічну по шагову гру “Танки”.

Сутність гри полягає в тому щоб одна з двох ворогуючих команд вивела зі строю юнітів ворогів. Можливості гри:

1)Кількість гравців – 2;

2)Зміна сторони двобою по чергова;

3)Кожен з гравців має по 2 середні танки і по 3 легкі;

4)Управління виконуватиметься мишею та клавіатурою;

5)Існуватиме прицільний вистріл - права клавіша миші, і ліва для пострілу;

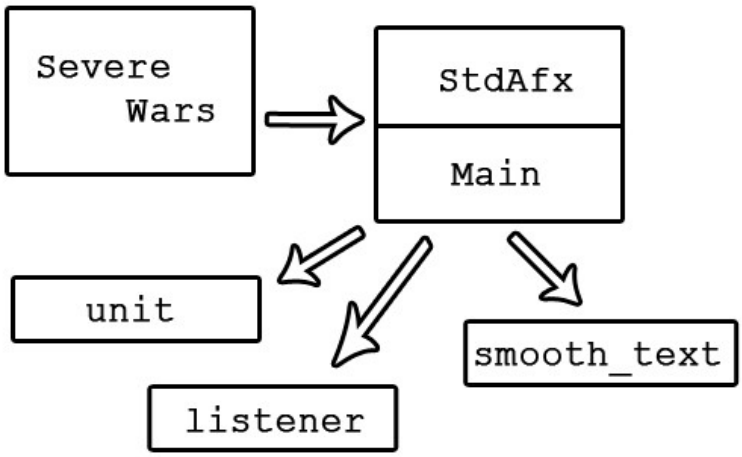
6)Лінійка життя буде змінюватися динамічно в залежності від нанесених ушкоджень;

7)З кінцем гри на екран монітора виводиться повідомлення про ту команду яка виграла;

8)Зміна графічних опцій.

2. Опис функціональних можливостей

Локальна структура гри



Main.cpp

«Ядро» гри. Використовує та пов’язує усі класи і функції між собою.

StdAfx.h

Також існує для пов’язання усіх файлів проекту. Вирішує питання з областю визначення змінних. Включає в себе важливі константи.

```
#include <iostream>
```

Бібліотека **iostream** яка керує вводом-виводом, як і **stdio.h**

```
#include <cmath>
```

Бібліотека алгебраїчних функцій.

```
#include <string>
```

Бібліотека роботи з нуль-термінованими рядками і різними функціями роботи з пам'яттю.

```
using namespace std;
```

```
#include <windows.h>
```

Бібліотека являє собою Windows-специфічний заголовний файл для програмування C.

```
#include <stdlib.h>
```

Бібліотека яка містить в собі функції, що займаються виділенням пам'яті, контроль процесу виконання програми, перетворення типів та інші.

```
#include <malloc.h>
```

Бібліотека динамічного розподілу пам'яті(спосіб виділення оперативної пам'яті комп'ютера для об'єктів у програмі, при якому виділення пам'яті під об'єкт здійснюється під час виконання програми)

```
#include <memory.h>
```

Дана бібліотека призначена для автоматичного виділення / звільнення пам'яті. Вонареалізована у вигляді шаблонного класу Memory.

```
#include <tchar.h>
```

За допомогою бібліотеки **Tchar** можна створювати додатки, що

використовують однобайтових кодування, багатобайтове кодування або кодування Unicode, з одних вихідних файлів.

```
#include <time.h>
```

Бібліотека, що містить типи і функції для роботи з датою й часом.

```
#include <vector>
```

Це шаблон зі стандартної бібліотеки C + +, який реалізує динамічний масив (контейнер std :: vector в C + +).

Для підгруження Xors 3d:

```
#include <xors3d.h>
```

Для підгруження:

```
#include "FIXorsBridge.h"
```

```
#include "unit.h"
```

```
#include "listener.h"
```

Для плавного відображення тексту:

```
#include "smooth_text.h"
```

Константи:

```
static char buffer[256];
```

Для роботи зі строками

```
static FILE *file;
```

Для роботи з файлами

```
static const int COURSE_SECONDS=30;
```

Тривалість ходу

```
static const int LANGUAGE_FROM=21;
static const int LANGUAGE_TO=13;
```

Кількість стовпчиків в файлі “language.ini”

Управління:

```
static const float CAMERA_SMOOTHNESS=.1f;
```

Плавність стеження камери за юнітом.

```
static const float HOUSING_SMOOTHNESS=.0250f;
```

Змінні камери:

```
static int camera;
static int camera_active_pivot;
```

Клас unit (unit.h, unit.cpp)

Відповідає за створення, оновлення та визволення танків.

```
class unit
{
private:
...
        bool ussr;
```
Найменування команди юніта.
```
        bool destroyed;
```
Зничення танка.

Для імітації фізики:
```
        float speed;
```
Швидкість танка.

```
        float TANK_BACK_SPEED_OFFSET;
```
Швидкість зміщення танку в зворотньому напрямі.

```
        float TANK_BACK_MAX_SPEED;
```
Максімальна швидкість зворотнього руху танка.

```
        float TANK_SPEED_OFFSET;
```
Швидкість зміщення танка.

```
        float TANK_MAX_SPEED;
```
Максимальна швидкість танка.

```
        float turning;
```
Поворот танка.

```
        float TANK_TURNING_OFFSET;
```
Зміщення танка при повороті.

```
        float TANK_MAX_TURNING;
```
Максимальна кількість градусів поворота танка.

```
        float MOUSE_FACTOR;
```
Фактор руху миші.

```
        ...
public:
        ...
        int type;
```
Ця змінна визначає тип танку.

```
        int mesh;
```
Визначає 3d модель танку.

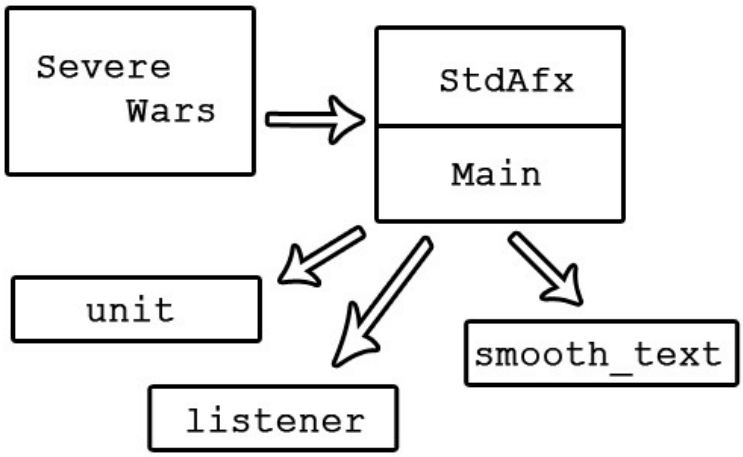
```
        string name;
```
Найменування танка.

```
        int life;
```
Тривалість життя.
```
        ...
```

int TANK_CAMERA_Y;
Змінна віддаленості камери від юніту Y.

int TANK_CAMERA_Z;
Змінна віддаленості камери від юніту X.

...
};




```
bool LoadSettings();
void CritError(int);
bool LoadLanguage();
void ShowMessage(bool, const char *, const char *);
//
```

Клас unit (unit.h, unit.cpp)

Відповідає за створення, оновлення та визволення танків.

```
class unit
{
private:
```

...
bool ussr;
Найменування команди юніта.
bool destroyed;
Зничення танка.

Для імітації фізики:
float speed;
Швидкість танка.

float TANK_BACK_SPEED_OFFSET;
Швидкість зміщення танку в зворотньому напрямі.

float TANK_BACK_MAX_SPEED;
Максімальна швидкість зворотнього руху танка.

float TANK_SPEED_OFFSET;
Швидкість зміщення танка.

float TANK_MAX_SPEED;
Максимальна швидкість танка.

```
        float turning;
```
Поворот танка.

```
        float TANK_TURNING_OFFSET;
```
Зміщення танка при повороті.

```
        float TANK_MAX_TURNING;
```
Максимальна кількість градусів поворота танка.

```
        float MOUSE_FACTOR;
```
Фактор руху миші.

```
        ...
public:
        ...
        int type;
```
Ця змінна визначає тип танку.

```
        int mesh;
```
Визначає 3d модель танку.

```
        string name;
```
Найменування танка.

```
        int life;
```
Тривалість життя.
```
        ...
        int TANK_CAMERA_Y;
```
Змінна віддаленості камери від юніту Y.

```
        int TANK_CAMERA_Z;
```
Змінна віддаленості камери від юніту X.

```
        ...
};
```

listener.cpp

Подієва система таймерів. Використовується для зворотного відліку та деяких затримань в ігровому процесі.

```
class listener
{
private:
        ...
        int time;
```
Тривалість.
```
        ...
};
```

smooth_text.cpp

Система плавного відображення тексту в стовпчик. Використовує "listener.cpp".

```
class smooth_text
{
```

private:
vector<string> str;
Відповідає за текст;

vector<int> x;
Координати тексту по осі x;
vector<int> y;
Координати тексту по осі y;

vector<int> font;
Шрифт тексту;

vector<float> transparent;
Прозорість.
};

Ігрові функції

Main.cpp

bool LoadSettings()

Завантажує ігрові налаштування з файлу “settings.cfg”. У випадку вдалого завантаження – повертає true, в іншому випадку - false.

bool LoadLanguage()

Завантажує текст гри з файлу “language.ini”. У випадку вдалого завантаження – повертає true, в іншому випадку - false.

void ShowMessage(const char *)

Виводить повідомлення в певному форматі.

Клас unit (unit.h, unit.cpp)

void initialize();

1) Завантаження моделі у відеопам’ять
2) Настройка фізики в залежності від типу танку

void update();

1) Отримання координат швидкості руху миші
2) Повертає башту танку та камеру
3) Робить прицілювання
4) Обмежує башту та дуло
5) Отримує інформацію про нажаті клавіши
6) Переміщує танк

Клас listener (listener.h, listener.cpp)

void set(int t);

Встановлює тривалість таймеру дії.

bool update();

Оновлює таймер. Функція повертає **true** тільки у випадку, якщо дія відбулася.

Клас smooth_text (smooth_text.h, smooth_text.cpp)

void add(string, int);

Додає стовпець тексту. (Другий параметр – шрифт тексту.)

bool draw(int, int);

Виводить текст в стовпчик по заданим координатам.

3. Розробка алгоритму

4. Лістинг програми

```
#include "StdAfx.h"

int APIENTRY WinMain(HINSTANCE instance, HINSTANCE prevInstance, LPSTR commandLine,
int commandShow)
{
	if (LoadSettings()==false) CritError(2);

	xGraphics3D(GRAPHICS_WIDTH, GRAPHICS_HEIGHT, GRAPHICS_DEPTH, GRAPHICS_MODE,
false);
	xSetTextureFiltering(TF_ANISOTROPICX4);
	xAntiAlias(true);
	xSetAntiAliasType(xGetMaxAntiAlias());
	xHidePointer();

	if (LoadLanguage()==false) CritError(3);

	FI_RENDER = cFIXorsBridge::Initiate();
	if (!FI_RENDER) CritError(4);

	FI_font[0]=FI_RENDER->LoadImageFont("data/b52_16.txt");
	FI_font[1]=FI_RENDER->LoadImageFont("data/b52_32.txt");

	// "Загружается... Подождите"
	ShowMessage(false, language[0].c_str(), language[1].c_str());

	camera = xCreateCamera();
	xCameraClsColor(camera, 203, 230, 255);
	xCameraRange(camera, 0.1f, 100000.0f);
	xCameraFogMode(camera, 1);
	xCameraFogColor(camera, 55, 55, 55);
	xCameraFogRange(camera, 500, 1000);

	xAmbientLight(135, 140, 135);

	int light = xCreateLight();
	//xRotateEntity(light, 90, 0, 0);
	xPositionEntity(light, 35, 30, 0);

	int light_visible=xCreateSphere(32, light);
	xEntityColor(light_visible, 0, 0, 255);
	xScaleEntity(light_visible, 15, 15, 15);

	int terrain = xCreateCube();//xLoadTerrain("data/media/height_map.bmp");
	xScaleEntity(terrain, 512.0f, 0.001f, 512.0f);
	xEntityColor(terrain, 30, 100, 0);
	//xMoveEntity(terrain, -256.0f, 0, -256.0f);
	//xTerrainDetail(terrain, 4096);
	//int grass = xLoadTexture("data/media/ground.jpg");
	//xScaleTexture(grass, 0.01f, 0.01f);
	//xEntityTexture(terrain, grass, 0, 0);
	//xEntityFX(terrain, 1);

	//listener tmrCourse(COURSE_SECONDS);

	bool course=false; // Какая команда ходит?
	int course_1=1; // Какой танк первой команды ходит?
	int course_2=1; // Какой танк второй команды ходит?
	int active_unit=course_1; // Какой танк активен сейчас?

	// Cell Shade
```

```
	//toon_shader=xLoadFXFile("data/ToonShaded Light.fx");
	//toon_shader_texture=xLoadTexture("data/ToonGradient.png");
	//

	// Иннициализация юнитов
	const int UNITS_QUANTITY=10; // Количество юнитов
	unit units[UNITS_QUANTITY+1];
	int units_i=1; // Переменная для перебора юнитов
	//

	int location=xLoadAnimMesh("data/media/location.b3d");
	xScaleEntity(location, 0.064f, 0.064f, 0.064f);

	int rs_c=0;
	double rs_x[5], rs_y[5], rs_z[5];
	int rm_c=0;
	double rm_x[2], rm_y[2], rm_z[2];
	int gs_c=0;
	double gs_x[5], gs_y[5], gs_z[5];
	int gm_c=0;
	double gm_x[2], gm_y[2], gm_z[2];

	int i, t_child;
	static const char *t_name;
	for(i=0; i<=xCountChildren(location)-1; i++)
	{
		t_child=xGetChild(location, i);
		t_name=xEntityName(t_child);
		if (strcmp(t_name, "Tree")==0) xEntityType(t_child,
4);//xEntityColor(t_child, 0, 0, 0);
		if (strcmp(t_name, "rs")==0)
		{
			rs_c++;
			rs_x[rs_c]=xEntityX(t_child, true);
			rs_y[rs_c]=xEntityY(t_child, true);
			rs_z[rs_c]=xEntityZ(t_child, true);
			sprintf(buffer, "ussr - s");
			units[rs_c].initialize(true, buffer, 0);
			xPositionEntity(units[rs_c].mesh, rs_x[rs_c], rs_y[rs_c],
rs_z[rs_c]);
			units[rs_c].seton(terrain);
		}
		if (strcmp(t_name, "rm")==0)
		{
			rm_c++;
			rm_x[rm_c]=xEntityX(t_child, true);
			rm_y[rm_c]=xEntityY(t_child, true);
			rm_z[rm_c]=xEntityZ(t_child, true);
			sprintf(buffer, "ussr - m");
			units[3+rm_c].initialize(true, buffer, 1);
			xPositionEntity(units[3+rm_c].mesh, rm_x[rm_c], rm_y[rm_c],
rm_z[rm_c]);
			units[3+rm_c].seton(terrain);
		}
		if (strcmp(t_name, "gs")==0)
		{
			gs_c++;
			gs_x[gs_c]=xEntityX(t_child, true);
			gs_y[gs_c]=xEntityY(t_child, true);
			gs_z[gs_c]=xEntityZ(t_child, true);
			sprintf(buffer, "german - s");
			units[5+gs_c].initialize(false, buffer, 0);
			xPositionEntity(units[5+gs_c].mesh, gs_x[gs_c], gs_y[gs_c],
```

```
gs_z[gs_c]);
                  units[5+gs_c].seton(terrain);
            }
            if (strcmp(t_name, "gm")==0)
            {
                  gm_c++;
                  gm_x[gm_c]=xEntityX(t_child, true);
                  gm_y[gm_c]=xEntityY(t_child, true);
                  gm_z[gm_c]=xEntityZ(t_child, true);
                  sprintf(buffer, "german - m");
                  units[8+gm_c].initialize(false, buffer, 1);
                  xPositionEntity(units[8+gm_c].mesh, gm_x[gm_c], gm_y[gm_c],
gm_z[gm_c]);
                  units[8+gm_c].seton(terrain);
            }
      }

      for(int i=0; i<=xCountChildren(location)-1; i++)
      {
            t_child=xGetChild(location, i);
            t_name=xEntityName(t_child);
            if (strcmp(t_name, "rs")==0 || strcmp(t_name, "rm")==0 || strcmp(t_name,
"gs")==0 || strcmp(t_name, "gm")==0)
            {
                  xHideEntity(xGetChild(location, i));
            } else {
                  xEntityType(t_child, 4);
            }
      }
      xCollisions(1, 4, SPHERETOTRIMESH, RESPONSE_SLIDING);
      xCollisions(3, 4, SPHERETOTRIMESH, RESPONSE_STOP);

      xPositionEntity(camera, xEntityX(units[active_unit].mesh),
xEntityY(units[active_unit].mesh), xEntityZ(units[active_unit].mesh));
      xEntityParent(camera, units[active_unit].mesh);
      xMoveEntity(camera, 0.0f, (float)units[active_unit].TANK_CAMERA_Y, -
(float)units[active_unit].TANK_CAMERA_Z);
      xEntityParent(camera, 0);
      xPointEntity(camera, units[active_unit].mesh);

      smooth_text text;
      text.add("Does it work?", 25, 100, 1, 0.05f, FI_font[1]);
      text.add("Yes, it works!", 25, 125, 1, 0.05f, FI_font[1]);

//    ShowMessage(true, language[2].c_str(), language[3].c_str());

      //sprintf(buffer, "%f", delta_time);
      //ShowMessage(true, buffer, buffer);

      // Инициализация снарядов
      vector<int> projectile;
      vector<float> projectile_g; // Для имитации гравитации
      int projectile_i; // Для перебора снарядов
      int fire_sprite=xLoadSprite("data/media/fire_sprite.png", 1+2);
      xEntityFX(fire_sprite, 1);
      xScaleSprite(fire_sprite, 0.5, 0.5);
      xHideEntity(fire_sprite);
      bool shooted=false;
      int shooted_t=0;
      //

      bool next_course=false;
      bool nc=false;
```

```
	int nc_t=0;

	// Игровой цикл
	while(!xKeyDown(KEY_ESCAPE))
	{
		Sleep(10);
		// Смена хода
		// tmrCourse.update()==true ||
		if (xKeyHit(KEY_SPACE) || next_course==true)
		{
			//if(clock()>tmrCourseEnd || xKeyHit(KEY_SPACE)) {
			//tmrCourseEnd = clock () + COURSE_SECONDS * CLK_TCK;
			//tmrCourse.set(COURSE_SECONDS);
/*			if (course==true)
			{
				course=false;
			} else {
				course=true;
			}
*/
			next_course=false;
			if (course==true)
			{
				course_1++;
				if (course_1>=6) course_1=1;
				active_unit=course_1;
				//if (active_unit==2) return 0;
			} else {
				course_2++;
				if (course_2>=6) course_2=1;
				active_unit=5+course_2;
			}
			course=!course;
		}

		if (xMouseHit(1)) {
			projectile.push_back(xCreateSphere(6));
			xEntityColor(projectile.back(), 0, 0, 0);
			xScaleEntity(projectile.back(), 0.225f, 0.225f, 0.225f);
			xPositionEntity(projectile.back(),
xEntityX(units[active_unit].barrel_mesh, true),
xEntityY(units[active_unit].barrel_mesh, true),
xEntityZ(units[active_unit].barrel_mesh, true));
			xRotateEntity(projectile.back(),
xEntityPitch(units[active_unit].barrel_mesh, true),
xEntityYaw(units[active_unit].barrel_mesh, true),
xEntityRoll(units[active_unit].barrel_mesh, true));
			xMoveEntity(projectile.back(), 0, 0, 7.5);
			xEntityRadius(projectile.back(), 0.25);
			xEntityType(projectile.back(), 3);
			xCollisions(3, 1, SPHERETOSPHERE, RESPONSE_STOP);
			projectile_g.push_back(0.0);
			//
			xPositionEntity(fire_sprite,
xEntityX(units[active_unit].barrel_mesh, true),
xEntityY(units[active_unit].barrel_mesh, true),
xEntityZ(units[active_unit].barrel_mesh, true));
			xRotateEntity(fire_sprite,
xEntityPitch(units[active_unit].barrel_mesh, true),
xEntityYaw(units[active_unit].barrel_mesh, true),
xEntityRoll(units[active_unit].barrel_mesh, true));
			xMoveEntity(fire_sprite, 0, 0, 6);
			xShowEntity(fire_sprite);
```

```
			xEntityAlpha(fire_sprite, 1.0);
			shooted=true;
			shooted_t=0;
			//
		}

		//xPointEntity(light, units[active_unit].mesh);

		if (shooted==true)
		{
			if (shooted_t<100) {
				shooted_t+=20;
				xEntityAlpha(fire_sprite, 1.0f-shooted_t*0.01f);
			} else {
				xHideEntity(fire_sprite);
				shooted_t=0;
				shooted=false;
			}
		}

		camera_active_pivot=units[active_unit].housing_pivot_camera;
		if (xMouseDown(2))
		{
			camera_active_pivot=units[active_unit].barrel_pivot_camera;
		}
		units[active_unit].update(terrain); // Обновление активного юнита

		// Обновление снарядов
		for(projectile_i=0; projectile_i<(int)projectile.size(); projectile_i++)
		{
			if (xEntityCollided(projectile[projectile_i], 1))
			{
				for(units_i=1; units_i<=UNITS_QUANTITY; units_i++)
				{
					int
collided_ent=xCollisionEntity(projectile[projectile_i],
xCountCollisions(projectile[projectile_i])-1);
					if (collided_ent==units[units_i].mesh)
					{
						units[units_i].life-=25;
						if (units[units_i].life<=0) {
							//xSetEntityEffect(collided_ent, 0);
							units[units_i].destroy();
						}
						nc=true;
					}
				}
				xResetEntity(projectile[projectile_i]);
				xFreeEntity(projectile[projectile_i]);
				projectile_g.erase(projectile_g.begin()+projectile_i);
				projectile.erase(projectile.begin()+projectile_i);
				break;
			}
			if (xEntityCollided(projectile[projectile_i], 4))
			{
				xResetEntity(projectile[projectile_i]);
				xFreeEntity(projectile[projectile_i]);
				projectile_g.erase(projectile_g.begin()+projectile_i);
				projectile.erase(projectile.begin()+projectile_i);
				break;
			}
			xMoveEntity(projectile[projectile_i], 0, 0, 8);
			projectile_g[projectile_i]+=0.0075f;
```

```
                  xMoveEntity(projectile[projectile_i], 0,
-projectile_g[projectile_i], 0);
            }
            //

            if (nc==true)
            {
                  if (nc_t<50)
                  {
                        nc_t++;
                  } else {
                        nc_t=0;
                        nc=false;
                        next_course=true;
                  }
            }

            // Слежение камеры за активным юнитом
            if (camera_active_pivot==units[active_unit].barrel_pivot_camera)
            {
                  xPositionEntity(camera, xEntityX(camera_active_pivot, true),
xEntityY(camera_active_pivot, true), xEntityZ(camera_active_pivot, true));
                  xRotateEntity(camera, xEntityPitch(camera_active_pivot, true),
xEntityYaw(camera_active_pivot, true), xEntityRoll(camera_active_pivot, true));
            } else {
                  xTranslateEntity(camera, (float)((xEntityX(camera_active_pivot,
true)-xEntityX(camera, true))*CAMERA_SMOOTHNESS), (float)
((xEntityY(camera_active_pivot, true)-xEntityY(camera, true))*CAMERA_SMOOTHNESS),
(float)((xEntityZ(camera_active_pivot, true)-xEntityZ(camera,
true))*CAMERA_SMOOTHNESS));
                  xPointEntity(camera, units[active_unit].mesh);
            }
            //

            // Выравнивание танков в зависимости от рельефности ландшафта
            for(units_i=1; units_i<=UNITS_QUANTITY; units_i++)
            {
                  units[units_i].seton(terrain);
                  //xSetEffectVector(units[units_i].mesh, "PosLight",
xEntityX(light, true), xEntityY(light, true), xEntityZ(light, true));
                  //xSetEffectVector(units[units_i].mesh, "PosCam", xEntityX(camera,
true), xEntityY(camera, true), xEntityZ(camera, true));
            }
            //

            xKeyDown(KEY_F2)==1?xWireframe(true):xWireframe(false);
            xUpdateWorld();
            xRenderWorld();

            // Отрисовка 2D графики
            FI_RENDER->StartDraw();
                  FI_RENDER->SetBlend(FI_ALPHABLEND);
                  FI_RENDER->SetAlpha(1.0f);
                  FI_RENDER->SetImageFont(FI_font[1]);

                  sprintf(buffer, "TrisRendered: %i; FPS: %i", xTrisRendered(),
xGetFPS());
                  FI_RENDER->DrawTextA(buffer, 10*(int)X_OFFSET, 10*(int)Y_OFFSET);

                  // Вывод информации о юнитах
                  FI_RENDER->SetImageFont(FI_font[0]);
                  for(units_i=1; units_i<=UNITS_QUANTITY; units_i++)
                  {
```

```
                        xCameraProject(camera, xEntityX(units[units_i].mesh),
xEntityY(units[units_i].mesh), xEntityZ(units[units_i].mesh));
                        if (units[units_i].isgood())
                        {
                              xColor(255, 0, 0);
                        } else {
                              xColor(0, 0, 255);
                        }
                        xText(xProjectedX()-
xStringWidth(units[units_i].name.c_str())/2, xProjectedY()-120,
units[units_i].name.c_str());
                        xColor(0, 0, 0);
                        xRect(xProjectedX()-52, xProjectedY()-77, 104, 6, true);
                        xColor(255, 0, 0);
                        xRect(xProjectedX()-50, xProjectedY()-75,
units[units_i].life, 2, true);
                  }
                  text.draw();
            FI_RENDER->EndDraw();
            xFlip();
      }
      return 0;
}

bool LoadSettings()
{
      string temp_str[2];
      file=fopen("settings.cfg", "r");
      if (!file) {
            return false;
      }
      else {
            if (fgets(buffer, 100, file)!=NULL) puts (buffer);
            if (fgets(buffer, 100, file)!=NULL) puts (buffer);
            temp_str[0]=buffer;
            temp_str[0]=temp_str[0].substr(0, temp_str[0].length()-1);
            if (fgets(buffer, 100, file)!=NULL) puts (buffer);
            temp_str[1]=buffer;
            temp_str[1]=temp_str[1].substr(0, temp_str[1].length()-1);
            if (temp_str[0]=="1280" & temp_str[1]=="1024") {
                  GRAPHICS_WIDTH=1280;
                  GRAPHICS_HEIGHT=1024;
            } else if (temp_str[0]=="1024" & temp_str[1]=="768") {
                  GRAPHICS_WIDTH=1024;
                  GRAPHICS_HEIGHT=768;
            } else if (temp_str[0]=="1024" & temp_str[1]=="768") {
                  GRAPHICS_WIDTH=1024;
                  GRAPHICS_HEIGHT=768;
            } else if (temp_str[0]=="800" & temp_str[1]=="600") {
                  GRAPHICS_WIDTH=800;
                  GRAPHICS_HEIGHT=600;
            } else if (temp_str[0]=="640" & temp_str[1]=="480") {
                  GRAPHICS_WIDTH=640;
                  GRAPHICS_HEIGHT=480;
            }
            if (fgets(buffer, 100, file)!=NULL) puts (buffer);
            temp_str[0]=buffer;
            temp_str[0]=temp_str[0].substr(0, temp_str[0].length()-1);
            if (temp_str[0]=="32") GRAPHICS_DEPTH=32;
            if (temp_str[0]=="16") GRAPHICS_DEPTH=16;
            if (temp_str[0]=="8") GRAPHICS_DEPTH=8;
            if (fgets(buffer, 100, file)!=NULL) puts (buffer);
            temp_str[0]=buffer;
```

```
            temp_str[0]=temp_str[0].substr(0, temp_str[0].length()-1);
            if (temp_str[0]=="full") {
                  GRAPHICS_MODE=true;
            } else {
                  GRAPHICS_MODE=false;
            }
            fclose(file);
            X_OFFSET=(1.0f/800.0f)*(float)GRAPHICS_WIDTH;
            Y_OFFSET=(1.0f/600.0f)*(float)GRAPHICS_HEIGHT;
            return true;
      }
}

bool LoadLanguage()
{
      file=fopen("language.ini", "r");
      if (!file) {
            return false;
      }
      else {
            for(int j=0; j<LANGUAGE_FROM+LANGUAGE_TO; j++) {
                  if (fgets(buffer, 100, file)!=NULL) puts (buffer);
                  if (j==0) {
                        language[j]=buffer;
                        language[j]=language[j].substr(0,language[j].length()-1);
                        xAppTitle(language[j].c_str());
                  }
                  if (j>=LANGUAGE_FROM) {
                        language[j-LANGUAGE_FROM]=buffer;
                        language[j-LANGUAGE_FROM]=language[j-
LANGUAGE_FROM].substr(0,language[j-LANGUAGE_FROM].length()-1);
                  }
            }
            fclose(file);
            return true;
      }
}

void ShowMessage(bool wait, const char* s1="", const char* s2="")
{
      xCls();
      FI_RENDER->StartDraw();
            FI_RENDER->SetBlend(FI_ALPHABLEND);
            FI_RENDER->SetRotation(0);
            FI_RENDER->SetImageFont(FI_font[1]);
            FI_RENDER->SetColor(255, 255, 255);
            FI_RENDER->DrawTextA(s1, (GRAPHICS_WIDTH/2-FI_RENDER-
>StringWidth(s1)/2), (GRAPHICS_HEIGHT/2-FI_RENDER->StringHeight(s1)/2));
            FI_RENDER->DrawTextA(s2, (GRAPHICS_WIDTH/2-FI_RENDER-
>StringWidth(s2)/2), (GRAPHICS_HEIGHT/2-(FI_RENDER->StringHeight(s2)*-2)/2));
      FI_RENDER->EndDraw();
      xFlip();
      if (wait==true) {
            xFlushKeys();
            xWaitKey();
            xFlushKeys();
            xMoveMouse(GRAPHICS_WIDTH/2, GRAPHICS_HEIGHT/2);
            xFlushMouse();
      }
}

void CritError(int id)
{
```

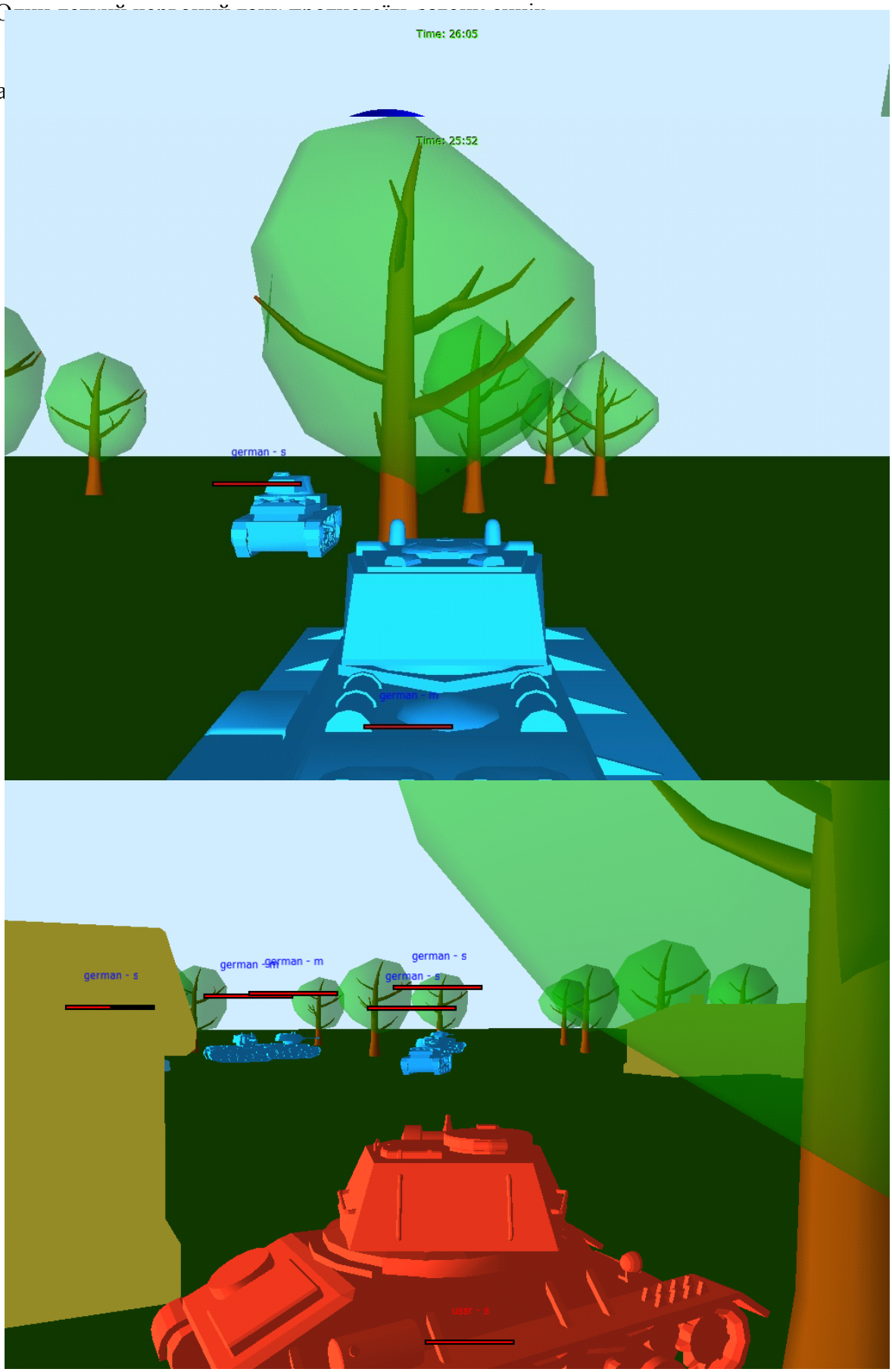
```
/*
2 - settings loading
3 - language loading
4 - FastImage initialization
*/
switch(id)
{
	case 1:
		break;
	default:
		MessageBox(NULL, _T(""), _T("Error"), MB_OK+MB_ICONERROR);
}
exit(1);
                                        }
```

5. Тестування програми

Мал. 1. На ньому зображено синю команду, а перед ними цілий загін червоних. Синя сфера Майбутнє джерело світла.

Мал. 2. О

Ма



6. Результаті реалізації

В результаті виконаної роботи нам вдалось створити тривимірний світ в якому є тривимірні моделі локації, і юнітів. Ми встигли зробити:

1)прицілювання;
2)постріл;
3)переміщення танків;
4)смугу життя над кожним із юнітів;
5)вихід із строю підбитого юніта;
6)завершальний напис перемоги чи поразки;
7)вибір графічних опцій;
8)зміна між сторонами двобою.

В майбутньому хотіли би зробити:

1)збереження і завантаження рівня;
2)покращити графічну і тривимірну медію;
3)покращити кількісний склад ігрових локацій;
4)головне меню;
5)ефекти;
6)звукове супроводження і музичне;
7)мережеві баталії;
8)таблиця рекордів;
9)власний сайт стратегії;
10)штучний інтелект для гри наодинці із комп’ютером;
11)ігрову кар’єру;

Список використаної літератури

Язык программирования C++. Спец. изд. Новое **Бьярне Страуструп (2011)**

C++ для начинающих Шаг за шагом (изд.2011 г.) **Герберт Шилдт**

Язык программирования C++. Спец. изд. Новое **Бьярне Страуструп (2011)**

Висновок

В результаті виконання цієї курсової роботи я здобув велику кількість знань в роботі із тривимірними об’єктами, ознайомився із можливостями графічного двигуна Xors 3d, навчився працювати із бібліотеками Direct X, отримав практичні навички в створенні мелії для гри, а саме створення динамічних і статичних тривимірних об’єктів ігрової сцени, познайомився із специфічними бібліотеками які потрібні для обміну даними між комп’ютером и програмою в реальному часі, здобув чималі знання що до відладки і виправлення помилок у власній програмі.