南开大学 计算机大类

姓名 王亦辉

学号 2310764

班级 工科试验班1-3

2024年5月14日

高级语言程序设计

实验报告

目录

[高级语言程序设计大作业实验报告 2](#_Toc166620037)

[**一.** **作业题目** 2](#_Toc166620038)

[**二.** **开发软件** 2](#_Toc166620039)

[**三.** **课题要求** 2](#_Toc166620040)

[**四.** **主要流程** 2](#_Toc166620041)

[**五.** **收获** 4](#_Toc166620042)

高级语言程序设计大作业实验报告

1. **作业题目**

一款无双割草小游戏

1. **开发软件**

Visual Studio 2022

使用了EasyX绘图库

1. **课题要求**
2. 面向对象。
3. 通过函数与类减少代码重复。
4. 使游戏尽可能流畅。
5. **主要流程**
6. **实现思路：**

通过定义玩家，敌人，图集，动画等类，使用类的封装特性来将复杂的事物封装成一个整体，方便使用。从游戏的数据读取，处理数据，绘制画面这三个大方向出发分别进行相应流程的编写。将游戏分成两个部分：开始界面与游戏进行界面分别处理。使用EasyX库中的ExMessage类对象进行输入操作的读取，得到ExMessage类对象后针对不同的目的分别编写相应函数来处理信息，从而进行游戏画面的绘制。比如，在玩家类中，

class Player

{

public:

const int FRAME\_WIDTH = 80;

const int FRAME\_HEIGHT = 80;

public:

Player()

{

loadimage(&img\_shadow, \_T("img/shadow\_player.png"));

anim\_left = new Animation(atlas\_player\_left,45);

anim\_right = new Animation(atlas\_player\_right,45);

}

~Player()

{

delete anim\_left;

delete anim\_right;

}

void ProcessEvent(const ExMessage&msg)

{

switch(msg.message)

{

case WM\_KEYDOWN:

switch (msg.vkcode)

{

case VK\_UP:

is\_move\_up = true;

break;

case VK\_DOWN:

is\_move\_down = true;

break;

case VK\_LEFT:

is\_move\_left = true;

break;

case VK\_RIGHT:

is\_move\_right = true;

break;

}

break;

case WM\_KEYUP:

switch (msg.vkcode)

{

case VK\_UP:

is\_move\_up = false;

break;

case VK\_DOWN:

is\_move\_down = false;

break;

case VK\_LEFT:

is\_move\_left = false;

break;

case VK\_RIGHT:

is\_move\_right = false;

break;

}

break;

}

}

void Move()

{

int dir\_x = is\_move\_right - is\_move\_left;

int dir\_y = is\_move\_down - is\_move\_up;

double len\_dir = sqrt(dir\_x \* dir\_x + dir\_y \* dir\_y);

if (len\_dir != 0)

{

double normalized\_x = dir\_x / len\_dir;

double normalized\_y = dir\_y / len\_dir;

position.x += (int)(SPEED \* normalized\_x);

position.y += (int)(SPEED \* normalized\_y);

}

if (position.x < 0)position.x = 0;

if (position.y < 0)position.y = 0;

if (position.x + FRAME\_WIDTH > WINDOW\_WIDTH)position.x = WINDOW\_WIDTH - FRAME\_WIDTH;

if (position.y + FRAME\_HEIGHT > WINDOW\_HEIGHT)position.y = WINDOW\_HEIGHT - FRAME\_HEIGHT;

}

void Draw(int delta)

{

int pos\_shadow\_x = position.x + (FRAME\_WIDTH / 2 - SHADOW\_WIDTH / 2);

int pos\_shadow\_y = position.y + FRAME\_HEIGHT - 8;

putimage\_alpha(pos\_shadow\_x, pos\_shadow\_y, &img\_shadow);

static bool facing\_left = false;

int dir\_x = is\_move\_right - is\_move\_left;

if (dir\_x < 0)

facing\_left = true;

else if (dir\_x > 0)

facing\_left = false;

if (facing\_left)

anim\_left->Play(position.x, position.y, delta);

else

anim\_right->Play(position.x, position.y, delta);

}

const POINT& GetPosion()const

{

return position;

}

private:

const int SHADOW\_WIDTH = 32;

const int SPEED = 3;

private:

Animation\* anim\_left;

Animation\* anim\_right;

IMAGE img\_shadow;

POINT position = { 500,500 };

bool is\_move\_up = false;

bool is\_move\_down = false;

bool is\_move\_left = false;

bool is\_move\_right = false;

};

通过ProcessEvent函数来对接收到的输入进行处理，然后move函数可以更新玩家的位置，而通过draw来在画面中绘制玩家。

1. **使用到的技巧：**
2. 运用GetTickCount函数来获得每次循环的运行时间,由视觉残留知识,1秒钟内绘制画面一定次数就可以让游戏显得连贯，从而我们可以不必频繁运行循环,通过sleep来让循环暂停,从而大幅减少cpu占用.
3. 运用渲染缓冲区，通过BeginBatchDraw，FlushBatchDraw，FlushBatchDraw函数以及cleardevice函数,使绘制先在缓冲区进行再将缓冲区切换到显示区域，从而使绘制过程不可见，使得画面不会闪烁
4. 运用享元的设计模式，只需在最开始加载图片材质，运行过程中新建enemy对象时不需要再从磁盘中读取图片材质，从而加快游戏主循环运行速度，减少游戏的卡顿感。比如：

class Animation

{

public:

Animation(Atlas\* atlas,int interval)

{

anim\_atlas = atlas;

interval\_ms = interval;

}

~Animation() = default;

void Play(int x, int y, int delta)

{

timer += delta;

if (timer >= interval\_ms)

{

idx\_frame = (idx\_frame + 1) % anim\_atlas->frame\_list.size();

timer = 0;

}

putimage\_alpha(x, y, anim\_atlas->frame\_list[idx\_frame]);

}

private:

int timer = 0;

int idx\_frame = 0;

int interval\_ms = 0;

private:

Atlas\* anim\_atlas;

};

通过使用Atlas类型的指针，需要同一批图片作为其动画的物品就可以指向同一个Atlas对象，从而不必为每一个物品单独加载图片，而只需要加载一次，在绘制时用同一个素材绘制多遍即可。

1. 通过将位移向量化，来保持玩家和敌人沿各个方向移动时速度相同，而不会在沿斜线走时速度变成正常的倍。
2. **收获**

初步学习了游戏的制作流程，设计模式。对C++语言有了更进一步的了解，对类的封装特性有了更深的体会，学会使用类来表现某一种事物的特性与行为，并使代码组织得更有逻辑性，不显得混乱。