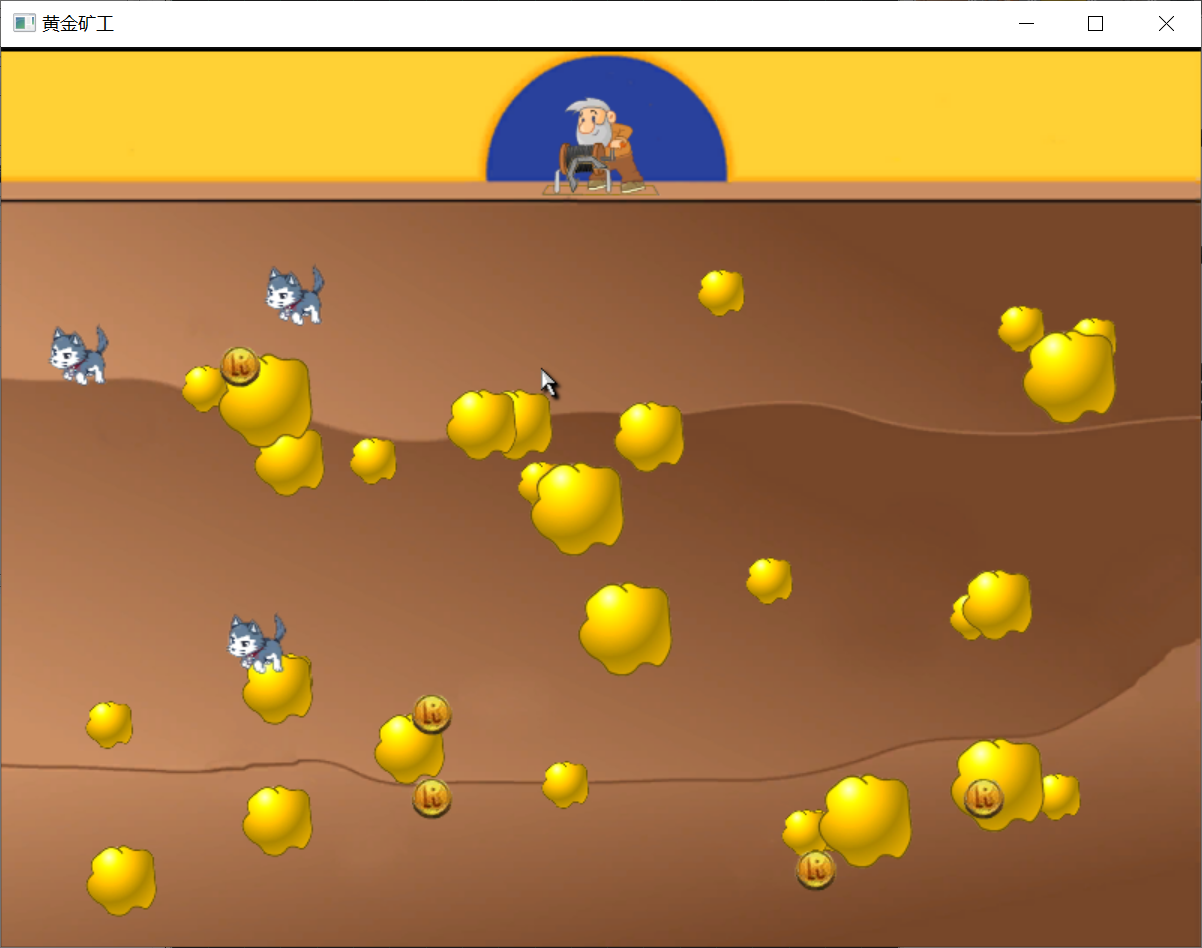
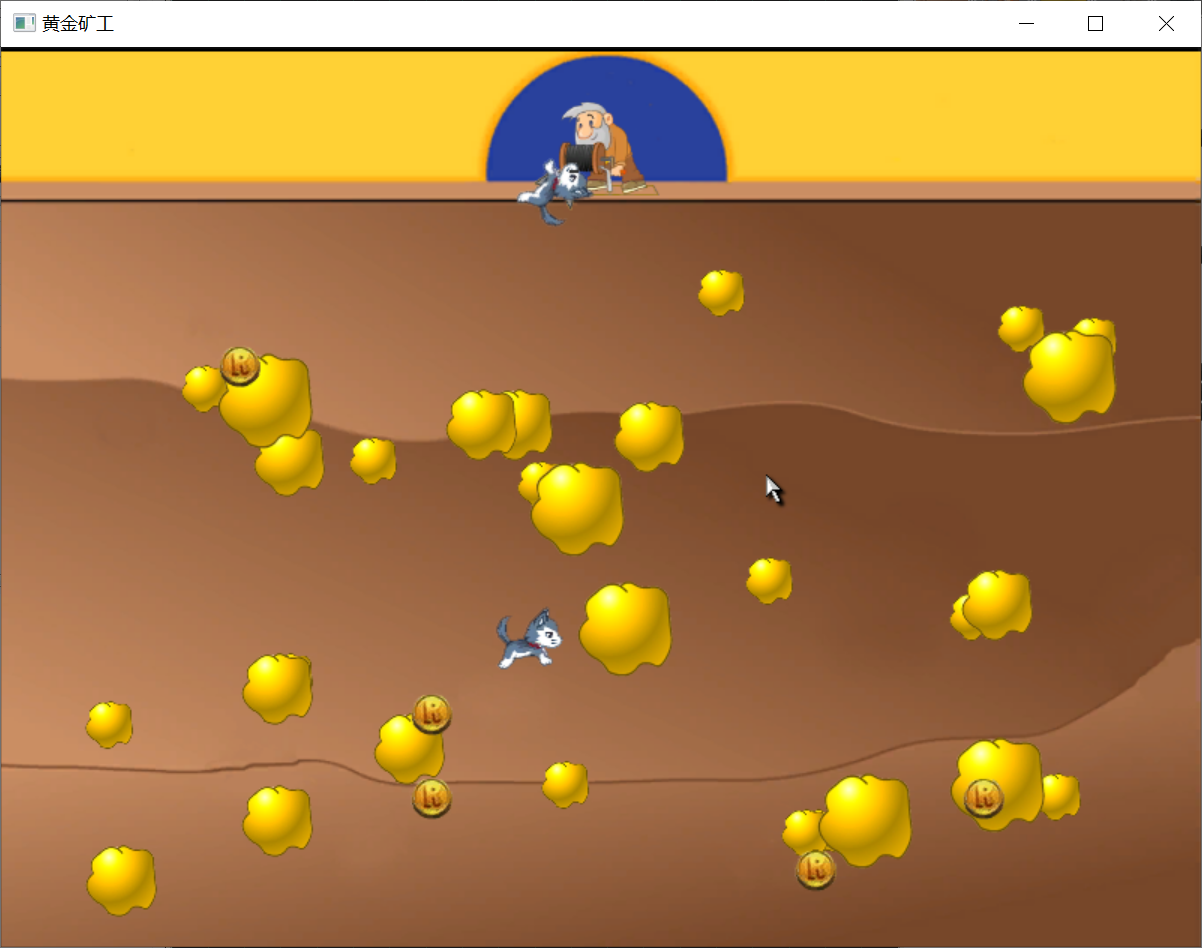
“海底世界”实验1报告

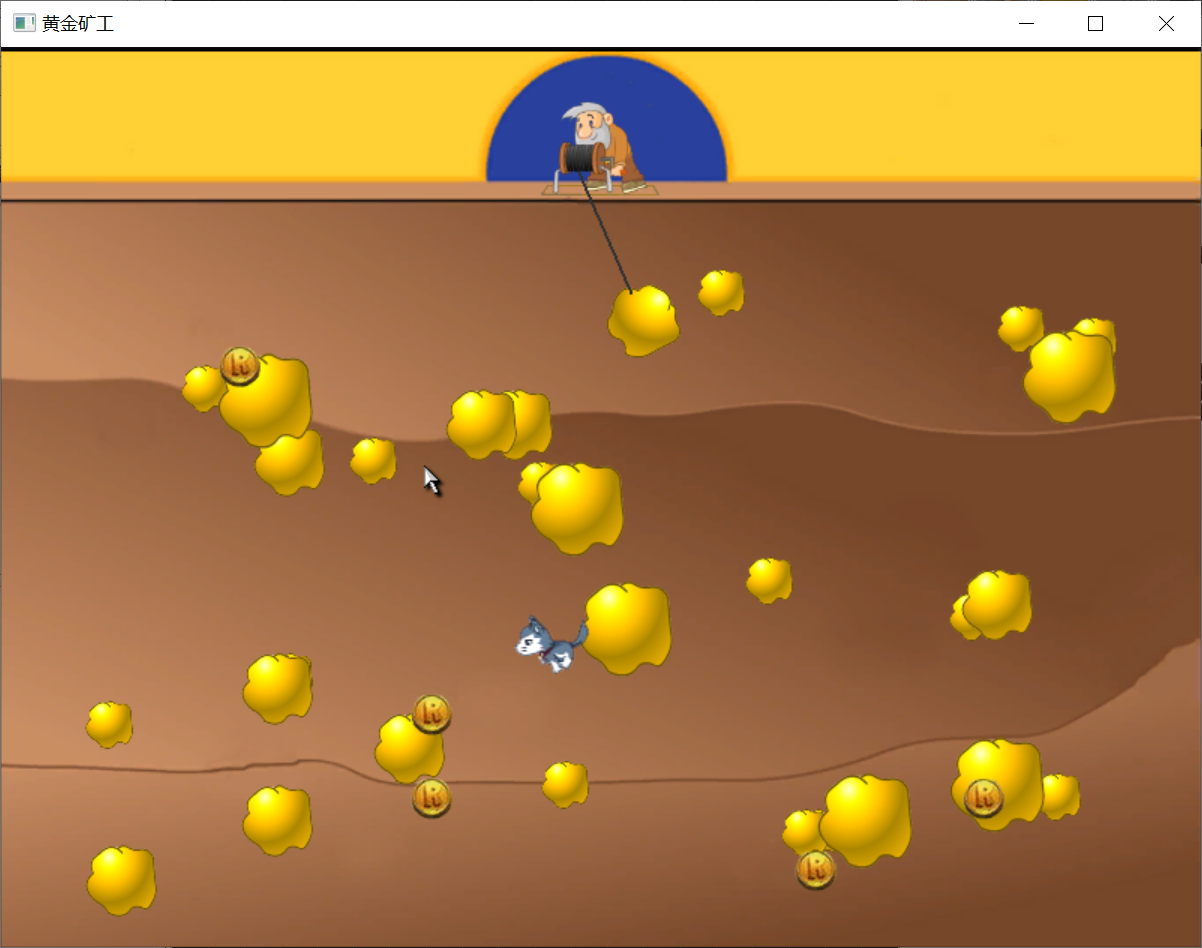
## 游戏截图



图一：进入游戏



图二：按键盘down键，抓住小狗



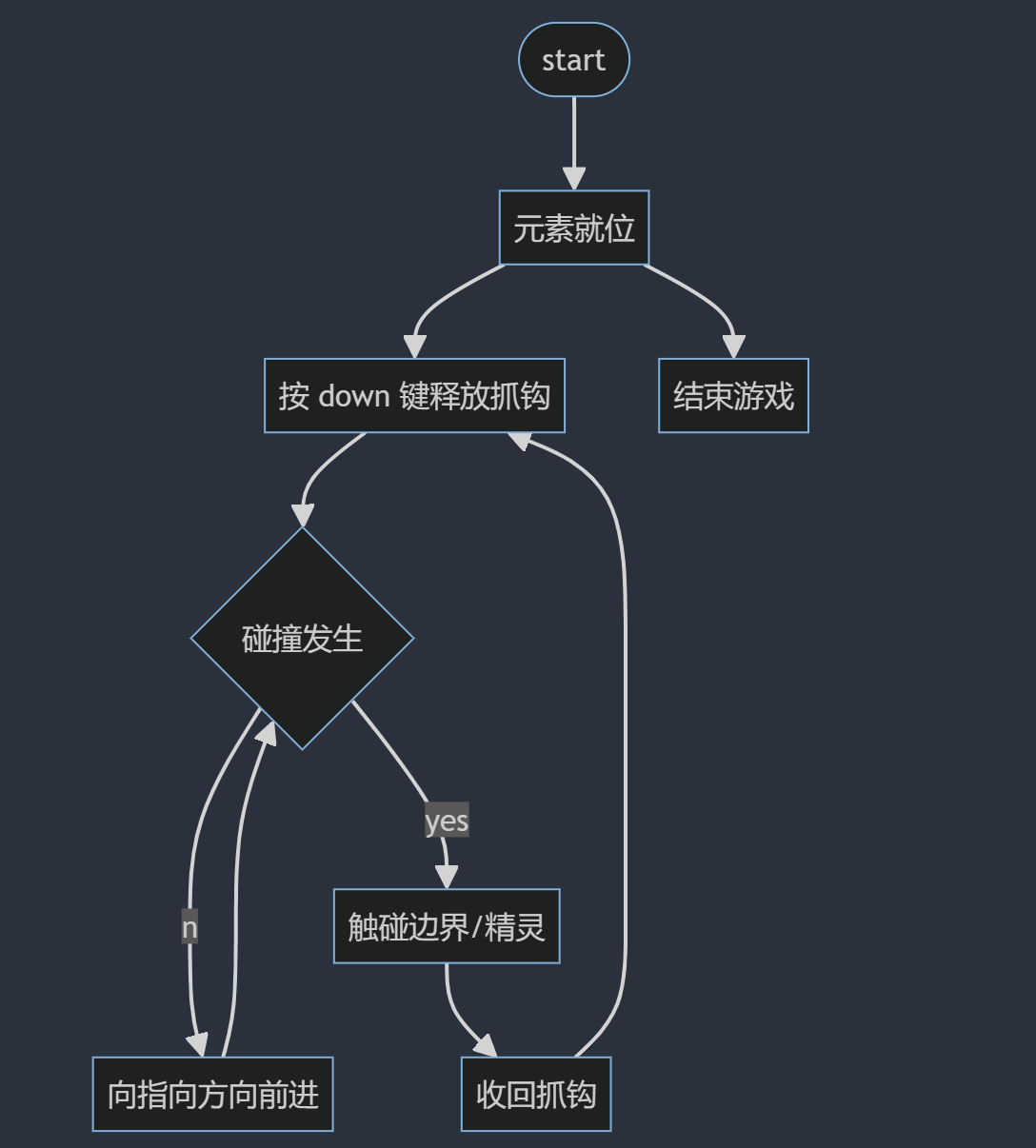
图三：抓住金子

## 功能描述

游戏名为黄金矿工

1. 游戏开始，生成金子、小狗和金币，抓钩来回摆动。
2. 玩家通过键盘down键可以控制抓钩下放。
3. 游戏开始后，用户按下相应的按键操控抓钩，抓取金子、金币和小狗。
4. 关闭窗口后即为游戏结束。重新启动将重复游戏过程。

## 流程图



## 数据结构

1. int m\_iGameState; // 游戏状态，0：结束或者等待开始；1：初始化；2：游戏进行中
2. int m\_fGoldBornMinX;
3. int m\_fGoldBornMaxX;
4. int m\_fGoldBornMinY;
5. int m\_fGoldBornMaxY;
6. int m\_iGoldCount;
7. vector<CSprite\*>golds; //管理金块，原理同海底世界中的rockFishs类似
8. int m\_iGoldCoinCount;
9. vector<CSprite\*>goldCoins; //管理金币，原理同上
10. int m\_iDogCount;
11. vector<CSprite\*>Dogs;
12. float m\_fHookRotation; //钩子同地面的夹角
13. int m\_iHookRotToLeft; //钩子摆动的方向：1 ← ；0 →
14. CSprite\* goldHook; //钩子对应的精灵
15. int m\_iHookState; //0 表示钩子旋转，1表示释放绳索，钩子不旋转,2表示回拉过程
16. float m\_fHookSpeed; //钩子释放的速度
17. CAnimateSprite\* goldMan; //矿工对应的Sprite
18. float m\_fHookStartPosX; //存储钩子的初始位置
19. float m\_fHookStartPosY;
20. CSprite\* m\_gotGold; //存储抓到的金块

*CGameMain::CGameMain()构造函数*

*完成初始化工作*

1. 设置游戏状态为1。
2. 初始化精灵goldHook、goldMan

*CGameMain::GameMainLoop(float fDeltaTime )*

*游戏的主循环，根据游戏的状态切换逻辑*

1. 当游戏状态为1时，调用GameInit()方法初始化游戏，然后置游戏状态为2。
2. 当游戏状态为2时，调用GameRun(fDeltaTime)方法，抓钩被控制着移动释放。游戏结束时会执行GameEnd且置状态为0。

*CGameMain::GameInit()*

*被抓取矿物的初始化*

*void GameMainLoop( float fDeltaTime );*

*void GameInit();*

*void GameRun( float fDeltaTime );*

*void GameEnd();*

*void OnKeyDown( const int iKey, const bool bAltPress, const bool bShiftPress, const bool bCtrlPress );*

*void OnSpriteColSprite( const char \*szSrcName, const char \*szTarName );*

*CSprite\* FindGoldSpriteByName(const char\* szName);*

*void OnSpriteColWorldLimit( const char \*szName, const int iColSide );*

*CSprite\* FindSpriteByName(const char\* szName);游戏开始和控制按钮*

1、如果任意键按下时，游戏未开始，即游戏状态为0，则将游戏状态改为1，运行游戏初始化函数（GameInit）

2、游戏开始后down按键可控制相应的抓钩释放。

3、所有的小狗精灵在碰到世界边界时都会反弹并反向。

4、可以抓住小狗、金块以及金币。

## 源代码

**LessionX.h：CGameMain类的声明**

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

//

//

//

//

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

#ifndef \_LESSON\_X\_H\_

#define \_LESSON\_X\_H\_

//

#include <Windows.h>

#include <vector>

using namespace std;

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

//

// 游戏总管类。负责处理游戏主循环、游戏初始化、结束等工作

class CGameMain

{

private:

int m\_iGameState; // 游戏状态，0：结束或者等待开始；1：初始化；2：游戏进行中

int m\_fGoldBornMinX;

int m\_fGoldBornMaxX;

int m\_fGoldBornMinY;

int m\_fGoldBornMaxY;

int m\_iGoldCount;

vector<CSprite\*>golds; //管理金块，原理同海底世界中的rockFishs类似

int m\_iGoldCoinCount;

vector<CSprite\*>goldCoins; //管理金币，原理同上

int m\_iDogCount;

vector<CSprite\*>Dogs;

float m\_fHookRotation; //钩子同地面的夹角

int m\_iHookRotToLeft; //钩子摆动的方向：1 ← ；0 →

CSprite\* goldHook; //钩子对应的精灵

int m\_iHookState; //0 表示钩子旋转，1表示释放绳索，钩子不旋转,2表示回拉过程

float m\_fHookSpeed; //钩子释放的速度

CAnimateSprite\* goldMan; //矿工对应的Sprite

float m\_fHookStartPosX; //存储钩子的初始位置

float m\_fHookStartPosY;

CSprite\* m\_gotGold; //存储抓到的金块

public:

CGameMain(); //构造函数

~CGameMain(); //析构函数

// Get方法

int GetGameState() { return m\_iGameState; }

// Set方法

void SetGameState( const int iState ) { m\_iGameState = iState; }

// 游戏主循环等

void GameMainLoop( float fDeltaTime );

void GameInit();

void GameRun( float fDeltaTime );

void GameEnd();

void OnKeyDown( const int iKey, const bool bAltPress, const bool bShiftPress, const bool bCtrlPress );

void OnSpriteColSprite( const char \*szSrcName, const char \*szTarName );

CSprite\* FindGoldSpriteByName(const char\* szName);

void OnSpriteColWorldLimit( const char \*szName, const int iColSide );

CSprite\* FindSpriteByName(const char\* szName);

};

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

//

extern CGameMain g\_GameMain;

#endif // \_LESSON\_X\_H\_

**LessonX：CGameMain类定义**

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

//

//

//

//

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

#include <Stdio.h>

#include <vector>

#include "CommonClass.h"

#include "LessonX.h"

#include <vld.h>

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

//

//

CGameMain g\_GameMain;

//==============================================================================

//

// 大体的程序流程为：GameMainLoop函数为主循环函数，在引擎每帧刷新屏幕图像之后，都会被调用一次。

//==============================================================================

//

// 构造函数

CGameMain::CGameMain()

{

\_CrtSetBreakAlloc(106);//

goldHook = new CSprite("goldhook"); //创建钩子实例

goldMan = new CAnimateSprite("goldMan");

m\_iGameState = 1;

}

//==============================================================================

//

// 析构函数

CGameMain::~CGameMain()

{

free(goldHook);

free(goldMan);

for (auto it : golds) {

free(it);

}

for (auto it : goldCoins) {

free(it);

}

for (auto it : Dogs) {

free(it);

}

}

//==============================================================================

//

// 游戏主循环，此函数将被不停的调用，引擎每刷新一次屏幕，此函数即被调用一次

// 用以处理游戏的开始、进行中、结束等各种状态.

// 函数参数fDeltaTime : 上次调用本函数到此次调用本函数的时间间隔，单位：秒

void CGameMain::GameMainLoop( float fDeltaTime )

{

switch( GetGameState() )

{

// 初始化游戏，清空上一局相关数据

case 1:

{

GameInit();

SetGameState(2); // 初始化之后，将游戏状态设置为进行中

}

break;

// 游戏进行中，处理各种游戏逻辑

case 2:

{

// TODO 修改此处游戏循环条件，完成正确游戏逻辑

if( true )

{

GameRun( fDeltaTime );

}

else // 游戏结束。调用游戏结算函数，并把游戏状态修改为结束状态

{

SetGameState(0);

GameEnd();

}

}

break;

// 游戏结束/等待按空格键开始

case 0:

default:

break;

};

}

//=============================================================================

//

// 每局开始前进行初始化，清空上一局相关数据

void CGameMain::GameInit()

{

m\_iGoldCount = 30; //金子初始数量

m\_iGoldCoinCount = 5; //金币数量

m\_iDogCount = 3; //狗数量

m\_fGoldBornMinX = CSystem::GetScreenLeft() + 5; //金子左边界

m\_fGoldBornMaxX = CSystem::GetScreenRight() - 5; //金子右边界

m\_fGoldBornMinY = CSystem::GetScreenTop() + 20; //金子上边界

m\_fGoldBornMaxY = CSystem::GetScreenBottom() -5; //金子下边界

m\_iHookRotToLeft = 1; //钩子初始化方向为←

m\_fHookRotation = 0.f; //初始化的夹角为0

m\_iHookState = 0; //初始情况下使钩子旋转

m\_fHookSpeed = 15.f;

m\_fHookStartPosX = goldHook->GetSpritePositionX(); //获取钩子的初始化X坐标

m\_fHookStartPosY = goldHook->GetSpritePositionY();//获取钩子的初始化Y坐标

int iLoop = 0; //循环变量控制

int iSize = 4, iPosX = 0, iPosY = 0;//iSize表示金块大小的变量

for( iLoop = 0; iLoop < m\_iGoldCount; iLoop++ ) {

if( iLoop < 15 ) //生成10个小金块，大小为4

{

iSize = 4;

}

else if( iLoop >= 15 && iLoop < 24 ) //生成6个中金块，大小为6

{

iSize = 6;

}

else //生成4个大金块，大小为8

{

iSize = 8;

}

//初始化金子精灵实例

char\* tmpName;

//生成金块名字

tmpName=CSystem::MakeSpriteName("GoldBlock",iLoop);

CSprite \*tmpSprite = new CSprite(tmpName);

tmpSprite->CloneSprite("goldTemplate");

tmpSprite->SetSpriteWidth(iSize); //设置金块的宽度

tmpSprite->SetSpriteHeight(iSize); //设置金块的高度

//随机的获取金子的X坐标和Y坐标

iPosX = CSystem::RandomRange(m\_fGoldBornMinX, m\_fGoldBornMaxX);

iPosY = CSystem::RandomRange(m\_fGoldBornMinY, m\_fGoldBornMaxY);

//设置金块精灵的位置

tmpSprite->SetSpritePosition(iPosX, iPosY);

golds.push\_back(tmpSprite); //将金块压入golds vector中集中管理

}

for( iLoop = 0; iLoop < m\_iGoldCoinCount; iLoop++ ) {

//初始化金币精灵实例

char\* tmpName;

//生成金币名字

tmpName=CSystem::MakeSpriteName("GoldCoin", iLoop);

CSprite\* tmpSprite = new CSprite(tmpName);

tmpSprite->CloneSprite("goldCoin");

//随机的获取金币的X坐标和Y坐标

iPosX = CSystem::RandomRange(m\_fGoldBornMinX, m\_fGoldBornMaxX);

iPosY = CSystem::RandomRange(m\_fGoldBornMinY, m\_fGoldBornMaxY);

//设置金币精灵的位置

tmpSprite->SetSpritePosition(iPosX, iPosY);

//将金币压入goldCoin vector中集中管理

goldCoins.push\_back(tmpSprite);

}

for( iLoop = 0; iLoop < m\_iDogCount; iLoop++ ){

//初始化狗精灵实例

char\* tmpName;

//生成狗名字

tmpName=CSystem::MakeSpriteName("Dog", iLoop);

CSprite\* tmpSprite = new CSprite(tmpName);

tmpSprite->CloneSprite("dog");

//随机的获取狗的X坐标和Y坐标

iPosX = CSystem::RandomRange(m\_fGoldBornMinX, m\_fGoldBornMaxX);

iPosY = CSystem::RandomRange(m\_fGoldBornMinY, m\_fGoldBornMaxY);

//设置狗精灵的位置

tmpSprite->SetSpritePosition(iPosX, iPosY);

//设置狗精灵向右移动

tmpSprite->SetSpriteLinearVelocity(-12, 0);

//将金币压入Dogs vector中集中管理

Dogs.push\_back(tmpSprite);

}

}

//=============================================================================

//

// 每局游戏进行中

void CGameMain::GameRun( float fDeltaTime )

{

if(m\_iHookState ==0){ //当m\_iHookState==0时钩子才摆动

const float fRotateSpeed = 45.f; // 摇摆速度，单位 度/秒

float fThisRotate = fRotateSpeed \* fDeltaTime; // 本次旋转的角度

if( m\_iHookRotToLeft ){ //向左转，度数不断变大

m\_fHookRotation += fThisRotate;

if( m\_fHookRotation >= 180.f ){ //大于180，置为向右转，即0

m\_fHookRotation = 180.f;

m\_iHookRotToLeft = 0;

}

}

else{ //向右转，度数不断变小

m\_fHookRotation -= fThisRotate; //小于0，置为向左转，即 1

if( m\_fHookRotation <= 0.f ){

m\_fHookRotation = 0.f;

m\_iHookRotToLeft = 1;

}

}

goldHook->SetSpriteRotation(m\_fHookRotation);//设置钩子的当前角度

}else if(m\_iHookState == 1 || m\_iHookState == 2){ //当钩子伸出后才需要画线

// 首先，从矿工精灵上获取一个缆绳链接点作为绳子的起始点(该链接点在编辑器里编辑好)

float fStartX = goldMan->GetSpriteLinkPointPosX(1);

float fStartY = goldMan->GetSpriteLinkPointPosY(1);

// 绳子终点在钩子精灵上获取(该链接点在编辑器里编辑好)

float fEndX = goldHook->GetSpriteLinkPointPosX(1);

float fEndY = goldHook->GetSpriteLinkPointPosY(1);

// 在这两点之间划线.线的颜色红绿蓝值都为50，即灰色；255表示不透明，2.0f表示线的粗细，0表示所在的层

CSystem::DrawLine(fStartX, fStartY, fEndX, fEndY, 2.f, 0, 50, 50, 50, 255 );

if(m\_iHookState == 2){ //回拉过程中

float fSpeedX = goldHook->GetSpriteLinearVelocityX(); //获取钩子X方向的速度

float fSpeedY = goldHook->GetSpriteLinearVelocityY(); //获取钩子Y方向的速度

//当速度接近为0时，即可判定其已到达初始点

if( fSpeedX < 0.00001f && fSpeedX > -0.00001f && fSpeedY < 0.00001f && fSpeedY > -0.00001f ){

m\_gotGold->SpriteDismount(); //解除金块与钩子的锚定

m\_gotGold->DeleteSprite(); //删除获取的金块

m\_iHookState = 0; //回拉结束，设定钩子状态为0,继续来回摆动

goldMan->AnimateSpritePlayAnimation("GolderManAnimation2", false); //播放矿工的动画，即准备拉金子的动画

}

}

}

}

//=============================================================================

//

// 本局游戏结束

void CGameMain::GameEnd()

{

}

void CGameMain::OnKeyDown( const int iKey, const bool bAltPress, const bool bShiftPress, const bool bCtrlPress ){

if( KEY\_DOWN == iKey && m\_iHookState == 0){ //按下↓并且游戏状态为 “1”

m\_iHookState = 1; //置游戏状态为1，可用于控制钩子是否摆动

// 以当前朝向给钩子一个向前的速度

goldHook->SetSpriteLinearVelocityPolar(m\_fHookSpeed, m\_fHookRotation);

// 播放挖金者的动作(一个胳膊往下压的动作)

goldMan->AnimateSpritePlayAnimation("GolderManAnimation1", 0); //0 表示播放一次，这里胳膊往下压就是一次

}

}

void CGameMain::OnSpriteColSprite( const char \*szSrcName, const char \*szTarName ){

if(strcmp(szSrcName, "goldhook") == 0 &&

(strstr(szTarName, "GoldBlock") !=NULL ||

strstr(szTarName, "Dog") !=NULL ||

strstr(szTarName, "GoldCoin") !=NULL

)){

m\_gotGold = FindGoldSpriteByName(szTarName); //在golds中查找是否存在对应的金块，并返回CSprite\*指针指向该金块

if(m\_gotGold!=NULL){

//思考：为什么链接点的位置为2？

m\_gotGold->SpriteMountToSpriteLinkPoint("GoldHook", 2 ); //将金块锚定在钩子上

goldHook->SpriteMoveTo(m\_fHookStartPosX, m\_fHookStartPosY, m\_fHookSpeed, 1); //使钩子向初始位置移动，即回拉

//思考：AnimateSpritePlayAnimation中的false参数的作用是什么？

goldMan->AnimateSpritePlayAnimation("GolderManAnimation3", false ); //播放拉金块的动作

m\_iHookState = 2; //表示金块回拉的状态

}

}

}

CSprite\* CGameMain::FindGoldSpriteByName(const char\* szName)

{

int iLoop;

for(int iLoop=0;iLoop<golds.size();iLoop++)

if(strcmp(szName,golds[iLoop]->GetName())==0)

return golds[iLoop];

for(int iLoop=0;iLoop<goldCoins.size();iLoop++)

if(strcmp(szName,goldCoins[iLoop]->GetName())==0)

return goldCoins[iLoop];

for(int iLoop=0;iLoop<Dogs.size();iLoop++)

if(strcmp(szName,Dogs[iLoop]->GetName())==0)

return Dogs[iLoop];

return NULL;

}

void CGameMain::OnSpriteColWorldLimit( const char \*szName, const int iColSide ){

CSprite\* tmp;

if(strcmp(szName,"goldhook") == 0){ //碰到边界的为钩子时

goldHook->SpriteMoveTo(m\_fHookStartPosX, m\_fHookStartPosY, m\_fHookSpeed, 1);

goldMan->AnimateSpritePlayAnimation("GolderManAnimation3", false );

m\_iHookState = 2; //同样设置为回拉，只是没有碰到金子

}

if((tmp = FindSpriteByName(szName))!=NULL){

if(iColSide == 1){

tmp->SetSpriteLinearVelocityX(-12.f);

tmp->SetSpriteFlipX(false);

}else if(iColSide == 0){

tmp->SetSpriteLinearVelocityX(12.f);

tmp->SetSpriteFlipX(true);

}

}

}

CSprite\* CGameMain::FindSpriteByName(const char\* szName)

{

for(int i=0;i<Dogs.size();i++) //size()返回vector的大小

if(strcmp(szName,Dogs[i]->GetName())==0) //遍历vector查找与szName具有相同的名称的Dogs中的精灵

return Dogs[i];

return NULL;

}

Main 类

//-----------------------------------------------------------------------------

//

//-----------------------------------------------------------------------------

#include "CommonClass.h"

#include "LessonX.h"

#include <vld.h>

#define \_CRTDBG\_MAP\_ALLOC

#include<stdlib.h>

#include<crtdbg.h>

//以下三行可省略无实际效果，在普通项目中可以

#ifdef \_DEBUG

#define new new(\_NORMAL\_BLOCK, \_\_FILE\_\_, \_\_LINE\_\_)

#endif

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

//

// 主函数入口

//

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

int PASCAL WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow)

{

// 初始化游戏引擎

if( !CSystem::InitGameEngine( hInstance, lpCmdLine ) )

return 0;

// To do : 在此使用API更改窗口标题

CSystem::SetWindowTitle("黄金矿工");

// 引擎主循环，处理屏幕图像刷新等工作

while( CSystem::EngineMainLoop() )

{

// 获取两次调用之间的时间差，传递给游戏逻辑处理

float fTimeDelta = CSystem::GetTimeDelta();

// 执行游戏主循环

g\_GameMain.GameMainLoop( fTimeDelta );

};

// 关闭游戏引擎

CSystem::ShutdownGameEngine();

return 0;

}

//==========================================================================

//

// 引擎捕捉鼠标移动消息后，将调用到本函数

void CSystem::OnMouseMove( const float fMouseX, const float fMouseY )

{

// 可以在此添加游戏需要的响应函数

}

//==========================================================================

//

// 引擎捕捉鼠标点击消息后，将调用到本函数

void CSystem::OnMouseClick( const int iMouseType, const float fMouseX, const float fMouseY )

{

// 可以在此添加游戏需要的响应函数

}

//==========================================================================

//

// 引擎捕捉鼠标弹起消息后，将调用到本函数

void CSystem::OnMouseUp( const int iMouseType, const float fMouseX, const float fMouseY )

{

// 可以在此添加游戏需要的响应函数

}

//==========================================================================

//

// 引擎捕捉键盘按下消息后，将调用到本函数

// bAltPress bShiftPress bCtrlPress 分别为判断Shift，Alt，Ctrl当前是否也处于按下状态。比如可以判断Ctrl+E组合键

void CSystem::OnKeyDown( const int iKey, const bool bAltPress, const bool bShiftPress, const bool bCtrlPress )

{

// 可以在此添加游戏需要的响应函数

g\_GameMain.OnKeyDown(iKey, bAltPress, bShiftPress, bCtrlPress );

}

//==========================================================================

//

// 引擎捕捉键盘弹起消息后，将调用到本函数

void CSystem::OnKeyUp( const int iKey )

{

// 可以在此添加游戏需要的响应函数

}

//===========================================================================

//

// 引擎捕捉到精灵与精灵碰撞之后，调用此函数

void CSystem::OnSpriteColSprite( const char \*szSrcName, const char \*szTarName )

{

g\_GameMain.OnSpriteColSprite(szSrcName, szTarName);

}

//===========================================================================

//

// 引擎捕捉到精灵与世界边界碰撞之后，调用此函数.

// iColSide : 0 左边，1 右边，2 上边，3 下边

void CSystem::OnSpriteColWorldLimit( const char \*szName, const int iColSide )

{

g\_GameMain.OnSpriteColWorldLimit(szName, iColSide);

}